Comprehensive Analysis on Secure Software Development Practices

Zaki Ahmad Rathore
Computer Science
University of Central Florida
Orlando, Florida, United States
za535945@ucf.edu

Shishir Singh
Computer Science
University of Central Florida
Orlando, Florida, United States
Shishir.singh@ucf.edu

Vigneshwar Sundararajan
Cybersecurity and Privacy
University of Central Florida
Orlando, Florida, United States
vi126747@ucf.edu

ABSTRACT

In today's interconnected world, where digital systems underpin almost every aspect of our lives, the security of software has never been more critical.

Data leaks, financial losses, and even threats to national security can result from software flaws. The techniques used by bad actors to exploit vulnerabilities evolve along with technology. Organizations must establish and continuously enhance safe software development processes to combat this changing threat landscape. In this proposal, a thorough study effort that will survey, examine, and improve secure software development processes across several businesses is described.

INTRODUCTION

The best strategy to maintain software is to always ensure security while the product is being developed. There are many factors that can influence security during the SDLC process, and we will discuss these phases next.

I. Requirement Analysis

In this phase, we need to make sure that we do threat modelling along with noting down all the security requirements as well. As much of it is documented, the better it is for all the stakeholders involved in the project. This also sets the right expectations from the start.

II. Design

We need to make sure that the designs are good and the cohesiveness between different objects and components of the software do not leave any unchecked vulnerability open to be attacked.

III. Implementation

We should also follow safe coding standards that emphasize security and also employ analysis on our code such as Static and Dynamic Analysis.

IV. Code Reviews and Testing

We should always go for manual code review before dynamic code review tools to verify intuitive errors and support accuracy of our tools. Code coverage is also important in our opinion.

V. Deployment and Maintenance

Deployment now a days is a big thing. We should make sure that we go for as much managed servers as possible. That is why cloud is a very secure option in comparison to spending the time to set up your own servers and managing all nodes and resources. There are many options in the cloud that take the guaranteed deployment and zero down time deployments which are good for secure and scalable applications. We should also make sure that our apps are regularly patched with the latest software updates and in the event of a new vulnerability, we can react to that fix very quickly.

PROBLEM STATEMENT

When we talk about Software Security, we must keep in mind that SDLC is a process that consists of many stages. Programming is not even half the total part of the life cycle in question. So, it is imperative to make sure that secure practices are followed along the way for most of the stages, if not all. The biggest problem in today's software development is the pace at which new frameworks, languages and tools are getting introduced.

With rapid pace of these tools being readily available, it is getting harder and harder to do your due diligence when it comes to vetting what is best for the company and the project. 30 years ago, it was hardly one or two main programming frameworks used for web development, while the introduction of so many tools and resources does help developers, it is often counterproductive for security.

RELATED WORK IN COMPLIANCE AND REGULATIONS

Modern software development has changed a lot. When it comes to new software, there are numerous new regulations and compliance policies introduced with them to ensure the security and privacy of its users and the software itself. Governments throughout the world

have developed legislation and compliance standards to set the bar for software development and data protection in the age of data breaches, cyberattacks, and growing privacy concerns.

I. **GDPR or General Data Protection Regulation** was enacted by the European union and mandates the software companies to make sure that there is no data breach of the user's data. There are also fail-safe mechanisms that need to be set within the software in case of a breach.

II. **CCPA or California Consumer Privacy Act** is a regulation by USA that is pivotal in providing rights to the consumers of goods and services. They give their personal information to these organizations and the organizations can not distribute their data without asking for consent.

III. **HIPAA** is also one of the biggest regulations out there that makes sure that private health information always stays private. No other doctor or human should be able to access anyone's medical records unless specifically authorized by the patient.

While these regulations are important, they also introduce a lot of complexity within the software development plan and limit innovation. That is why we must go above and beyond to incorporate practices that will not shadow our creativity and give us a better output.

TECHNIQUES

We have gone through a series of literature and surveys done on this topic and have found accurate input and things that work for modern software development.

I. **Software Development Life Cycle – SDLC** this blog introduces SDLC for us and breaks down all the phases that go into a development life cycle without having security as a concern. This article serves as a steppingstone towards other discoveries.

II. **The security development lifecycle by howard and lipner** presents microsofts security development lifecycle (SDLC) as a comprehensive framework for integrating security into software development processes. This paper advocates for the adoption of secure coding standards and emphasizes the importance of integrating security throughout the entire software development lifecycle.

III. **Classifying software changes clean or buggy? By kim, whitehead, and zhang** introduces a classification approach to distinguish between clean and buggy changes in software code, providing a means to identify potential vulnerabilities introduced during code modifications. This research sheds light on the importance of continuous monitoring and assessment of code changes.

IV. **Vccfinder: finding potential vulnerabilities in open-source projects to assist code audits proposes vccfinder**, a tool for automatically identifying potential vulnerabilities in open-source projects. This tool streamlines code audits and contributes to the reduction of security weaknesses in open-source software.

V. **An Approach for Secure Software Development Life Cycle in Small Software Firms by Prakash Chandra Behera** talks about the issues with development lifecyle and the security vulnerabilities that come with smaller working teams. It touches upon the topic of unrealistic deadlines and lack of training of new resources involved within this type of working model.

VI. **Secure Software Development Lifecycle: A case for Adoption in software SMES by Wisdom Umeugo** talks about the three principles of software security which are Confidentiality, Integrity and Availability. It also talks about malicious attacks in different phases of a software's life. They introduce various frameworks like Comprehensive Lightweight Application Security Process and Security Assurance Maturity Model (SAMM).

VII. **Security in the Software Development Lifecycle by Hala Assal and Sonia Chiasson** goes beyond the case study and conducts live interviews with developers to gather insightful data that involves software development. They also discuss how team size and other variables like project deadlines and software model (agile, waterfall etc.) influence the importance given to security in general.

VIII. **How to Manage Security & Third Party/Open-Source Code in the SDLC by Drew Brown** mentions the importance of security issues when using 3rd party tools or open-source code in your project. It talks about different types of scans like OS, IAST and DAST scans and how crucial they are when using open-source code to make sure there are no vulnerabilities in the version we are using.

IX. **Are Companies Actually Using Secure Development Life Cycles by David Geer** examines the use of secure development life cycles (SDLs) in major companies and startups. The article also discusses the trade-offs between cost and time in implementing secure lifecycles, noting how larger companies prioritize budget, while startups focus on time. It anticipates a rise in innovative and SaaS-based solutions to address these challenges.

X. **When a Patch Goes Bad: Exploring the Properties of Vulnerability-Contributing Commits** investigates the risks of vulnerable code in version control systems. Analyzing 68 vulnerabilities in the Apache HTTP Server, the study finds vulnerability-contributing commits are typically larger and more likely from new programmers.

It introduces metrics to mitigate these risks and critiques the current inefficiency of human code reviews in identifying vulnerabilities.

XI. **Predicting Vulnerable Software Components** is comprehensive research on vulnerable software components. They developed a tool called vulture and analyzed Mozilla's code with given vulnerability data for predictive analysis. The ML model built focused on vulnerable components and their root cause specifically focusing on buffer usage and functions. The paper concluded that the vulnerabilities in fact do have a strong correlation with patterns used by programmers, specifically method calls and imports (un-used or used).

XII. **Code Churn: A Measure for Estimating the Impact of Code Change by Sebastian G. Elbaum** introduces 'code churn' as a metric for assessing the complexity of software builds and the impact of code changes. The study emphasizes using this measure to understand relative complexity and guide software testing. It tells us that a code changed frequently is more likely to go vulnerable in the future.

XIII. **Socio-Technical Developer Networks: Should We Trust Our Measurements?** is a study evaluating the effectiveness of social network analysis (SNA) in software development. By surveying developers from the Linux kernel, PHP, and Wireshark projects and analyzing their experiences, the study validates the use of SNA metrics in understanding complex developer interactions. However, it advises treating these measurements as approximations rather than be all end all.

XIV. **Processes to Produce Secure Software** focuses on TSP which stands for Team Software Process for improving software development progress, particularly using secure methodologies. TSP helps developers streamline their goals with the team and withing their own personal growth umbrella as well. It also talks about learning, un-learning and relearning secure practices iteratively because of how fast software changes every day. It talks about the use of patches and the importance of not having any vulnerability within a version of a software you are using.

XV. **Biometric Security and Internet of Things (IoT)** talks about a very specific area within secure software, based on IOT. IOT is the future and its getting incorporated within our lives every day. It talks about how biometrics within IOT need to be top notch and should have 0 vulnerabilities to be able to take over traditional markets. It plans on authorization tiers and authentication protocols within different biometrics (face, fingerprint, eyes) and discusses what is covered within the spectrum of IOTs. The paper concludes with a discussion on the future scope of biometric security in IoT, considering technological advancements and emerging challenges.

XVI. **5G Internet of Things: A Survey** discusses the emergence of 5G networks and their increasing significance in the development of IoT applications are covered in the first section of the article. The software built within IOTs is supposed to be secure as well as fast and the paper discusses the intricacies of how the future of 5G aims to make 5G integration within smart IOT systems very seamless. It is relevant to software development security because technical challenges include scalability, network analysis, security, and privacy concerns. The software built for these systems needs to be secure enough to guarantee no outside access.

XVII. **Exploring software security approaches in software development lifecycle: A systematic mapping study** is a systematic mapping study analyzing security techniques in the SDLC. It reviews academic sources to identify 52 security approaches, categorizing them into areas like secure requirements modeling and vulnerability mitigation. The study finds a research gap in lifecycle-wide security checks and identifies key researchers and publication venues in this field, emphasizing early integration of security in the SDLC.

XVIII. **The Application of a New Secure Software Development Life Cycle (S-SDLC) with Agile Methodologies** examines how agile practices like SCRUM, XP, and Kanban can enhance software security. It critiques traditional agile methods for often overlooking security and suggests roles like software and security gurus. The paper advocates for varied meeting formats and the use of Wikibases and task backlogs, focusing on team practices rather than code.

XIX. **Social Influences on Secure Development Tool Adoption: Why Security Tools Spread by Shundan Xiao** is a case study exploring developers' perspectives on adopting security best practices and tools. The study dives into the varying importance of tools across teams, influenced by organizational factors and communication channels. It acknowledges that while cost is less of a concern for companies, time and other factors make widespread adoption challenging, offering realistic recommendations.

XX. **Secure Software Development Practice Adoption Model: A Delphi Study by Sri Lakshmi** examines secure software development in the Malaysian public sector, with relevance to other regions:

- Phase 1: Identification of factors influencing secure software development.
- Phase 2: Determination of assessment indicators for each identified factor.
- Phase 3: Mapping secure software development practices to the identified factors.

The study highlights the significant influence of project managers, developers, security experts, and security documentation in adopting secure software development techniques. It notes that rewards, incentives, and change management are less impactful. The study provides a framework for evaluating and improving secure software development in public sector organizations, aiding in decision-making and addressing negating factors.

XXI. **Evaluating JavaPathFinder on Log4J by David. A. Dickey** talks about JPF which is a model checker that investigates all potential execution pathways of a Java program to detect property violations. The authors evaluate JPF by putting it through six known flaws including uncaught runtime exceptions, unclosed resources, and threading difficulties.

XXII. **Secure Software Development Life Cycle Processes by N Davis** gives an overview of frameworks, and standards that support or could support secure software development. Namely CMMI, SSE-CMM FAA-iCMM, focusing on their relevance to programming practices. It also talks about Xtreme programming-pair programming and concludes with emphasizing on standardization.

XXIII. **Importance of Secure Software Development Processes and Tools for Developers by M. Danish** highlights the evolution of software from standalone, CD-based applications to internet-dependent systems and services. The study examines several SDLC security paradigms, such as Agile, Scrum, and Waterfall, among others. The need of adopting security at each stage of the SDLC to reduce vulnerabilities and threats is discussed.

XXIV. **What is the Secure Development Lifecycle (SDL)? On ptasecurity** talks about starting SDL easily. It highlights novel ideas like SDL Discovery, third party software tracking, static scanning in the development process and also highlights fuzzing, penetration testing and integration testing at the very end of every sprint cycle.

XXV. **Why is Dynamic Analysis an Important Part of Your AppSec Mix? By VeraCode** talks about their DAST dynamic analysis solution. It provides accurate and actionable results without a need for developer intervention saving time and resources. Most dynamic analysis done by DAST finish within the first 8 hours for biggest code bases and even less for smaller ones.

XXVI. **Risk Management Perspective in SDLC by Kavita Sahu** talks about risk management and analysis and it focuses on how risk analysis should be done prior to writing a single line of code. This makes sure that risk management is always relevant and never an afterthought.

XXVII. **About Some Methods for Software Security by Shafagat Mahmudova** talks about software tool access, implementing malware scans, ensuring software identification, and statistical analysis of software processes. The paper closes with a suggestion to incorporate encryption for software security and talks about the need of extensive research in private data processing.

XXVIII. **Information Security Requirement in Project Development Life Cycle authored by Manoj and Shabnam Sangwan** talks about interconnected apps focusing on large scale applications talking to each other. There are trust boundaries that need to be set up for secure communication and the paper highlights that these boundaries should be highly vetted before configuring to essential systems. The key takes away from this paper was the use of cryptography in security.

XXIX. **Software Quality Management: A Process to Improve the Quality of Software Project Using SQA by Chinmaya Dash, Dr. D Ramesh, and Prakash C Behera** talks about the importance of QA. It highlights the secure quality plan (SQP) for every software shipped and talks about the Software Quality Assurance lifecycle in detail. The key takes away from this paper is the relevance and ever highest demand for quality assurance not just in engineering but software development and how to achieve those QA compliance regulations.

XXX. **Technical and Personal Factors Influencing Developers' Adoption of Security Tools" by Jim Witschey, Shundan Xiao, and Emerson Murphy-Hill** talk about the personal interests of developers and the outside influence. It conducts a study and deduces that the background of a developer, level of experience and the company culture plays the highest role in influencing these decisions. It does also finish the study by emphasizing on a security first culture and embedding these values into new hires to make sure that the old habits do die, even if they take a long time.

EVALUATION/LITERATURE REVIEW

There are a lot of different papers and studies covered in this report and while not everything has been beneficial to our findings, there has been some good insight into security and software processes, and we will try to cover them using different correlated topics within the papers. The common themes discussed; common solutions discussed along with mitigation metrics**.**

I. Socio Technical Networks and IOT

While we analyze multiple papers, the disconnect between different new technologies, their adoption and their daily use is extreme when we go into the social construct of security. The papers in this correlation segment are [12][13][14][15]. These go into deep knowledge about 5G networks, the future of 5G networks, what is theoretically possible and what is realistic in terms of the economic conditions of the world.

Social Construct

This ignites an important yet hidden discussion about the social implications of highly secure systems and what it really takes to make them secure. There are a bunch of tools [14] discussed in this paper and recognizes the importance of their adoption. This, however, begs the question: how are we evaluating these tools? Do we have objective metrics [15] in place for these tools or are we just going to rely on subjective metrics that are highly flexible and fueled by intuition.

The human element in these intuitive analysis metrics can not be trusted and we need Social Network Analysis [15] as discussed in the 4th paper. Even with proper SNA, the study does conclude that there are gray areas in place, and we need to make sure we understand what we are getting ourselves into while implementing security and secure software development.

Influence with IOT

The reason why we believe these researches are related are not only because of the importance put on security but also because we can see that there is a delay in IOT adoption worldwide. There could be many reasons influencing that but one aspect is the economic scalability of nations, and the bigger reason is the fact that people do not trust their privacy while moving their smart homes to IOT.

Collectively, these papers at hand highlight the importance of emerging tech and the social context towards its adoption. IOT Infrastructure and security protocols for users is the topic revolved in these studies The social construct amongst these networks is a thing to be studied in the future among the ever going demand.

II. Software Vulnerability and its Impact

There are primarily three papers that come to our study correlating with each other in terms of vulnerability analysis and its impact on software security in the short term and in the longer more sustainable term. The studies [16][17][18] in these papers discuss common problems revolving around code commits and common procedures used within software teams. The study in the last paper [18] goes deep into what happens when a supposed fix causes more problems than it started with.

Vulnerable Code Analysis: A patch gone bad.

This paper focuses on identifying and analyzing coding mistakes that lead to issues in software, specifically in the Apache HTTP Server. The researchers used the term vulnerability commits for these commits in version control that contribute to vulnerabilities after the code is deployed.

Processes used in this study are related to other papers and in this study, a semi-automated method was used to find commits with vulnerabilities involving rigorous code review and fetching automated static and dynamic analysis tools.

Secure Software: Not only a buzz word

The process discussed in the 2nd paper are focused on team building and trying to find and solve problems as a unit with a proper methodology in place. TSP, however, is already discussed for this paper in the previous section so we can go past that for now. The 3rd paper focuses more on prediction and tries to counter the argument of retroactive analysis that happened in the 1st 2 papers here.

Common Code practices

While the code practices concluded in this research [18] were very detailed, they must be considered with a grain of salt as well. They go into the details about how import statements and over-use of modularity (method calls) can contribute to vulnerabilities found in common softwares. These are surprising to many industry experts as these are considered best-practices in many programming languages and frameworks. Again, this study did contribute to many more studies that followed this and the impact of not using much more methods than not using any has not been studied enough to form a very conclusive argument. For this literature review's scope, the patches that did go wrong are seen to be having a lot of these coding practices in place and supports the linearity of these arguments.

Conclusion

When coupled, the three studies provide a thorough understanding of security from several view points, including vulnerability prediction, development process security, and vulnerability analysis using empirical data. It is possible to use the knowledge obtained by examining VCCs and related churn metrics into the procedures used to create secure software. We can use these findings to develop something very comprehensive. This could not be just limited to security but also towards improving overall code quality as well.

III. Agile and Security

While many companies seem to follow the famous Agile software development lifecycle, the question about their security best practices remains a question needed to be answered. The papers discussed in this thread [19] [20] [21] dive deeper into answering these questions along with the future of these practices and their progression.

Secure Development Lifecycles

When discussing security, we need a framework so a lifecycle that has a step-by-step mitigation technique and a lot of reviewing going on is the best course of action. [19] goes into SDL adoption levels and discusses how there are only a handful of companies already using fool-proof or slight proof software engineering techniques when delivering software.

The research concludes that only 30.4% of the companies are using some sort of security protocols or have a check and balance within their code review process. The rest are mostly focused on pushing code out as soon as possible. The hypothesis in this study focuses on better developer tools in assisting engineers and predicts that the innovation in these tools is inevitable, which is a good sign.

Traditional vs Evolved Approach to SDL

The author [20] in this paper discusses traditionally secure techniques and the new ones being used widely now a days. It conducts the study on 52 techniques and maps them to different categories with a need to work on one category more than the other ones.

According to the report, security checks are typically considered during the coding stage of the SDLC, highlighting the need for more thorough coverage at every stage.

S-SDLC: A child name for Secure Agile

S-SDLC or Secure SDLC refers to integrating security best practices seamlessly into the current common SDLC techniques. It defines principles for implementing S-SDLC and roles like the security master and security gurus to integrate a security perspective within the development team.

Future of Security in Agile

The three articles emphasize the difficulties in incorporating security into the software development life cycle and offer ways to overcome them, such formal SDLs, grouping security techniques, or creating a new S-SDLC that is in line with agile techniques.

The third paper notably adds by discussing the integration of security into agile approaches, an issue not as extensively covered in the first two papers.

IV. Code churn metrics

The two studies in this category [22][23] are discussing the impact of different programming practices along with their possible impact. Code churns the keyword primarily refers to how much code needed to be changed to keep the software secure.

Impact and Findings

1. The papers highlight how code churn may play a significant role in the maintenance and development of software.
2. It talks about how a lot of code change might be sign of obvious flaws in the coding practice employed from the start.
3. The study also makes the case that keeping an eye on code churn might be useful in locating code segments that may be vulnerable to defects or security problems.

Adoption of churn metrics and their impact

The research analyzes several variables that affect the creation of safe software, including the institutional framework, individuals and their actions, project content, and system development procedures.

To establish safe software processes, it highlights the significance of security specialists, documentation, project management, and development methodology.

To improve software security, the paper [23] also covers the CLASP (Comprehensive Lightweight Application Security Process) paradigm and how it may be customized to meet the demands of certain organizations.

Intersection and Complementary Insights

Despite concentrating on distinct areas, both publications make a substantial contribution to the larger field of software security. The second paper's emphasis on secure development processes is related to the first paper's focus on code churn. Within a safe software development environment, code churn may be used as a measure to track and enhance program security. The second study highlights how secure software development approaches can be informed by the insights gained from monitoring code churn. For instance, more stringent security inspections or secure coding techniques may be needed in areas with high code churn.

Subsequent studies could examine in greater depth how code churn affects software system security and how code churn metrics can be included into safe software development models to increase their effectiveness in a range of organizational scenarios.

V. Secure Software in different SMEs

The three papers discussed in this area [1][2][5] are focused on corporate firms and small and medium sized at that. It is a good research group for our sake since this comprises of more than 80% of the companies or firms.

Key Takes

To reduce risks and vulnerabilities, all three papers stress how important it is to integrate security throughout the Software Development Life Cycle (SDLC).

The first paper [2], addresses the difficulties that SMEs encounter when implementing SSDLC, such as the idea that security is an extra feature, the expense and complexity of SSDLC, and the reliance on developers to provide security in the absence of sufficient knowledge.

The second paper [5], outlines the need for security to be a primary consideration throughout the SDLC, especially given the rise of cloud services and Internet-connected device.

In line with the second paper's recommendation [5] to use secure coding methods and security assessment tools, all papers provide a thorough overview of secure software frameworks, maturity models, and tools required for producing secure software.

Future Need with New Apps (Cloud, SaaS and Other)

A very important point discussed collectively in all three papers and the one which also correlates them with each other is the discussion of new technologies and the need for security.

With everything going to cloud-based including and not limited to the use of SaaS platforms along with deployment models, it is imperative to have fool proof pipelines for your code. These go beyond the traditional coding practices and is the reason why these

papers were the most thought provoking among every other paper we analyzed. Software security is now a lot more focused on secure and fast API calls, security keys, complex and thorough authentication, and authorization mechanisms along with the traditional best coding practices that we have already discussed within the previous papers.

Economic and Social Impact of Software Security:

The first study discusses the considerable economic impact of software security incidents [2]. This thing emphasizes the necessity of strong SSDLC procedures to prevent the high costs of fixing vulnerabilities and breaches.

This idea is reiterated in the last paper's conclusion, which emphasizes how important it is to create secure software for both social and economic advancement [1].

Concluding Literature Review

These papers highlight how crucial it is to include security at every stage of the SDLC, particularly when it comes to SMEs. They draw attention to the difficulties SMEs encounter when implementing the SSDLC, the importance of employing security-aware development techniques, and the financial consequences of software security. Future studies should investigate how to get SMEs to implement SSDLC and how to keep security procedures updated to fend off new threats.

DISCUSSION

This analysis also serves as a base to figure out how new techniques can enhance the works of these papers. The continuous progress of the software development landscape, along with emerging pain points, necessitates new approaches to secure development. By leveraging the findings from these papers, we can identify areas where innovation can lead to improved research outcomes:

I. Using S-SDLC instead of SDLC

We have come to this conclusion that we need to incorporate a security centric development lifecycle instead of a development or design specific one. Risk Management and assessment is something that needs to be considered every step of the way. We also need to incorporate penetration testing, integration testing and black box white box testing into our development lifecycle.

II. Use of Advanced Tools for Standardization and Concurrency

We have also analyzed a software called JPF (Java Path Finder) [31] into our research. We found that using tools like these immensely helps developers make sure that there are no surprises or missed conditional requirement violations post development. JPF helps set standardization rules and regulations for example: making sure no numerical value is going to exceed a given range. It also helps tackle concurrency control issues and solves major issues like Deadlocks and RACE conditions within the code. This can help immensely when taking software security seriously within your enterprise.

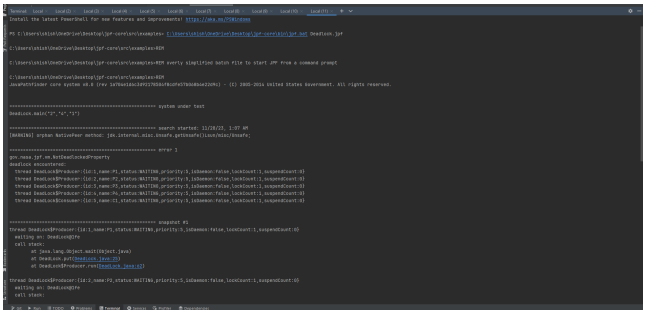

Figure 1: Deadlock Detection with JPF

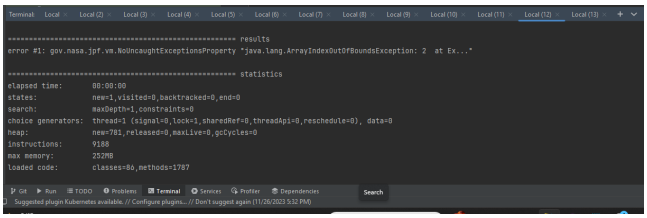

Figure 2: Numerical Value Check with JPF

III. Incorporating Machine Learning and AI

New techniques, such as the integration of machine learning and artificial intelligence, can bolster the classification system proposed by Kim et al. This can enable automated detection of potential vulnerabilities in code changes, improving the efficiency of their methodology.

IV. Security Automation and DevSecOps

Building on the foundations of the security development lifecycle, embracing devsecops principles and security automation can further streamline the integration of security controls throughout the software development process. This can create a more agile and responsive approach to security.

V. Advanced Code Analysis Tools

The research presented in vccfinder can be enhanced through the development of more advanced and accurate code analysis tools. The incorporation of advanced static and dynamic analysis techniques, along with improved vulnerability databases, can result in more effective vulnerability detection and mitigation.

VI. Implementing CI/CD Pipelines

One of the innovative techniques introduced is the implementation of CI/CD (continuous integration/ continuous deployment) pipelines, especially for code qualification with stringent security measures. This approach promises to enhance the existing coding

standards by automating security checks at every stage of development, thereby improving overall code quality and security.

VII. Holistic Security Integration

Beyond the immediate research findings, these innovations can infuse security throughout the entire software development lifecycle. They talk about not only the coding stage but also extend their influence to the critical phases of analysis, design, planning, implementation, and maintenance, thereby providing a proactive approach to software security that permeates every facet of the development process.

VIII. Preventive Measures in Every Phase

The application of innovative techniques should encompass a proactive stance in each phase of the software development lifecycle, fortifying the software against potential vulnerabilities and security threats before they arise. This approach shifts the focus from reactive measures to an ongoing, anticipatory strategy that safeguards the software from the very outset.

Figure 3: Implementation of Security in SDLC

This analysis aims to not only provide a comprehensive understanding of the existing research but also to highlight the potential for innovation and improvement in the field of secure software development practices. It serves as a foundation for exploring how new techniques and approaches can further advance the state of the art and contribute to more secure software development in a rapidly evolving threat landscape.

CONCLUSION AND RESULTS

In conclusion, this analysis has undertaken a thorough exploration of important research papers in the field of secure software development. We have picked apart their primary objectives, highlighting the significance of adopting secure coding standards, integrating security controls across the software development lifecycle, identifying prevalent security weaknesses, and the key role of security awareness and training programs.

Notably, we have investigated custom testing and evaluated tools and literature to find that there are modern tools and technologies already available to upskill and safeguard development. JPF [31] is only one example and we can use that to see how different concurrency issues and standardization regulations can be easily regulated. These techniques not only improve coding practices but also cast a protective net over the entire software development lifecycle, from initial analysis to ongoing maintenance.

Furthermore, we emphasized the importance of taking a proactive stance in every phase of software development, fortifying applications against potential vulnerabilities and security threats before they materialize. The insights from these research papers, coupled with the innovative techniques proposed involving cloud infrastructure, new tools like JPF and other modern software security techniques can help provide a roadmap for organizations and developers to navigate the intricacies of an ever-changing threat landscape.

WORK DISTRIBUTION

At the end of this project, we have completed the survey with all the papers we wished to complete. We went through every paper individually first and trying to cross reference ideas and techniques that can better the software security principles being used today.

The group member who led the project is Zaki Ahmad Rathore. We divided the papers in equal parts, and we have weekly checkpoint meetings in which we get together to discuss any finished papers and take notes about what we can include in our project report. Up until now, we have finished 31 papers and we have divided them into 11-10-10 denominations. Shishir and Vigneshwar completed their papers each and sent the findings to Zaki. Zaki completed the rest of the papers and wrote the final report. The report was then further discussed within the team for errors and possible corrections that could be done. We also configured and install JPF and it was collective effort among all three group members. The number of total hours spent individually are very hard to track but a good guess would be 100-120 hours for each group member involved.

REFERENCES

[1] Behera, P. C., Dash, C., & Yadav, S. K. (2021, May 03). An Approach for Secure Software Development Life Cycle in Small Software Firms. Indian Journal of Natural Sciences. Retrieved from https://www.researchgate.net/publication/353306538_An_Approach_for_Secure_Software_Development_Life_Cycle_in_Small_Software_Firms

[2] Umeugo, W. (2023). Secure Software Development Lifecycle: A Case for Adoption in Software SMEs. International Journal of Advanced Research in Computer Science. Retrieved from https://www.researchgate.net/publication/368737283_SECURE_SOFTWARE_DEVELOPMENT_LIFECYCLE_A_CASE_FOR_ADOPTION_IN_SOFTWARE_SMES

[3] Assal, H., & Chiasson, S. (2018, August 12–14). Security in the Software Development Lifecycle. The Advanced Computing Systems Association. Retrieved from https://www.usenix.org/system/files/conference/soups2018/soups2018-assal.pdf

[4] Davis, N. (2013, July). Secure Software Development Life Cycle Processes. Software Engineering Institute, Carnegie Mellon University. Retrieved from https://insights.sei.cmu.edu/documents/430/2013_019_001_297287.pdf

[5] Roshaidie, M. D., Liang, W. P. H., Jun, C. G. K., Yew, K. H., & Zahra, F.-t.-Z. (2020, December). Importance of Secure Software Development Processes and Tools for Developers. Taylor's University Selangor, Malaysia. Retrieved from https://www.researchgate.net/publication/348078670_Importance_of_Secure_Software_Development_Processes_and_Tools_for_Developers#fullTextFileContent

[6] Brown, D. (2019, August 15). How to Manage Security & Third Party/Open Source Code in the SDLC. CISO Platform. Retrieved from https://www.cisoplatform.com/profiles/blogs/how-to-manage-security-amp-third-party-open-source-code-in-the

[7] Software Development Life Cycle - SDLC. (2018, November 07). Retrieved from https://devqa.io/software-development-life-cycle-sdlc-phases/

[8] Howard, M., & Lipner, S. (2006, May). The Security Development Lifecycle. Microsoft Press. ISBN: 0735622140. Retrieved from https://www.researchgate.net/publication/234792172_The_Security_Development_Lifecycle

[9] Perl, H., Dechand, S., Smith, M., Arp, D., Yamaguchi, F., Rieck, K., Fahl, S., Acar, Y. (2015). VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits. Fraunhofer FKIE, Germany; University of Bonn, Germany; University of Göttingen, Germany; Saarland University, Germany. Retrieved from https://saschafahl.de/static/paper/vccfinder2015.pdf

[10] Chang, R.-Y., Podgurski, A., & Yang, J. (2008, September). Discovering Neglected Conditions in Software by Mining Dependence Graphs. IEEE Transactions on Software Engineering, 34. doi: 10.1109/TSE.2008.24. Retrieved from https://www.researchgate.net/publication/220070307_Discovering_Neglected_Conditions_in_Software_by_Mining_Dependence_Graphs

[11] Kim, S., Whitehead, E. J. Jr., & Zhang, Y. (2008). Classifying Software Changes: Clean or Buggy? IEEE Transactions on Software Engineering, 34(2). Retrieved from https://groups.csail.mit.edu/pag/pubs/kim-tse-2008.pdf

[12] Li, S., Xu, L. D., & Zhao, S. (2018, June). 5G Internet of Things: A survey. ScienceDirect, Journal of Industrial Information Integration. doi:10.1016/j.jii.2018.01.005. Retrieved from https://www.sciencedirect.com/science/article/abs/pii/S2452414X18300037?fr=RR-2&ref=pdf_download&rr=82ae4421a8a202dc

[13] Obaidat, M. S., Rana, S. P., Maitra, T., & Giri, D. (2018, October). Biometric Security and Internet of Things (IoT). ResearchGate. doi:10.1007/978-3-319-98734-7_19. Retrieved from https://www.researchgate.net/publication/328516514_Biometric_Security_and_Internet_of_Things_IoT

[14] Xiao, S., Witschey, J., & Murphy-Hill, E. (2014, February). Social influences on secure development tool adoption: why security tools spread. ACM. doi:10.1145/2531602.2531722. Retrieved from https://dl.acm.org/doi/abs/10.1145/2531602.2531722

[15] Meneely, A., & Williams, L. (2011, May 21). Socio-technical developer networks: should we trust our measurements? ACM. doi:10.1145/1985793.1985832. Retrieved from https://dl.acm.org/doi/abs/10.1145/1985793.1985832

[16] Neuhaus, S., Zimmermann, T., Holler, C., & Zeller, A. (2007, October). Predicting Vulnerable Software Components. ResearchGate. doi:10.1145/1315245.1315311. Retrieved from https://www.researchgate.net/publication/221609812_Predicting_Vulnerable_Software_Components

[17] Redwine, S. T. Jr., & Davis, N. (2004, March). Processes to Produce Secure Software. National Cyber Security Summit. Retrieved from https://www.cyberpartnership.org/Software%20Pro.pdf

[18] Meneely, A., Srinivasan, H., Musa, A., Rodríguez Tejeda, A., Mokary, M., & Spates, B. (2013, December 12). When a Patch Goes Bad: Exploring the Properties of Vulnerability-Contributing Commits. IEEE. doi:10.1109/ESEM.2013.19. Retrieved from https://ieeexplore.ieee.org/document/6681339

[19] Geer, D. (2010, June 07). Are Companies Actually Using Secure Development Life Cycles? IEEE. doi:10.1109/MC.2010.159. Retrieved from https://ieeexplore.ieee.org/document/5481927

[20] Mohammed, N. M., Niazi, M., Alshayeb, M., & Mahmood, S. (2017, February). Exploring software security approaches in software development lifecycle: A systematic mapping study. ScienceDirect, Computer Standards & Interfaces. doi:10.1016/j.csi.2016.10.001. Retrieved from https://www.sciencedirect.com/science/article/abs/pii/S0920548916301155?fr=RR-2&ref=pdf_download&rr=82ae444e1ae802dc

[21] De Vicente Mohino, J., Bermejo Higuera, J.-R., & Sicilia Montalvo, J. A. (2019, October). The Application of a New Secure Software Development Life Cycle (S-SDLC) with Agile Methodologies. ResearchGate. doi:10.3390/electronics8111218. Retrieved from https://www.researchgate.net/publication/336805102_The_Application_of_a_New_Secure_Software_Development_Life_Cycle_S-SDLC_with_Agile_Methodologies

[22] Munson, J.C., & Elbaum, S.G. (2002, August 06). Code churn: a measure for estimating the impact of code change. IEEE. doi:10.1109/ICSM.1998.738486. Retrieved from https://ieeexplore.ieee.org/document/738486

[23] Kanniahand, S. L., & Mahrin, M. N. b. (2018, July 5). Secure Software Development Practice Adoption Model: A Delphi Study. Journal of Telecommunication, Electronic and Computer Engineering. Retrieved from https://jtec.utem.edu.my/jtec/article/view/4462/3333

[24] Positive Research. (2020, February 25). How to approach secure software development. Retrieved from https://www.ptsecurity.com/ww-en/analytics/knowledge-base/how-to-approach-secure-software-development/

[25] Goslin, H. (2020, July 31). Why is Dynamic Analysis an Important Part of Your AppSec Mix? VERACODE. Retrieved from https://www-stage.veracode.com/blog/intro-appsec/why-dynamic-analysis-important-part-your-appsec-mix

[26] Kavita Sahu, Rajshree, Rajeev Kumar, "Risk Management Perspective in SDLC", International Journal of Advanced Research in Computer Science and Software Engineering, ISSN: 2277 128X, Volume 4, Issue 3, March 2014

[27] Shafagat Mahmudova, "Some Methods for Software Security", SOCIETY FOR SCIENCE AND EDUCATION UNITED KINGDOM, ISSN: 2054 -7420

[28] Manoj1, Shabnam Sangwan2. Information Security requirement in Project Development Life Cycle. International Journal of Advance Research in Computer Science and Management Studies. Volume 3, Issue 3, March 2015, ISSN: 232 7782 (Online)

[29] Chinmaya, D & Ramesh, D & Prakash CB, (2016). Software Quality Management: A Process to Improve the Quality of Software Project Using SQA. International Journal of Computer Science and Information Technologies, ISSN 0975 – 9646

[30] J. Witschey, S. Xiao, and E. Murphy-Hill. Technical and personal factors influencing developers' adoption of security tools. In ACM SIW, 2014.

[31] Dickey, D., Dorter, B., & German, J. (n.d.). Dickey Pathfinder - CMU School of Computer Science. https://www.cs.cmu.edu/~aldrich/courses/654-sp05/tools/dickey-pathfinder-05.pdf

[32] Citation: Figure1: https://intellisoft.io/what-is-a-secure-software-development-life-cycle-stages-methodologies-and-best-practices/