Towards Architectural Pen Test Case Generation and Attack Surface Analysis to Support Secure Design

Mahdi Jafari Sarvejahani
KASTEL - Institute of Information Security and Dependability
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
mahdi.jafari2@kit.edu

***Abstract*—In today's interconnected world, software systems have become indispensable components of complex frameworks, such as cyber-physical systems, e-commerce platforms, and healthcare information systems. This widespread integration highlights the importance of security more than ever, as these systems often function in critical environments where vulnerabilities can lead to significant destructive consequences. Penetration testing is a key method for identifying vulnerabilities but is often conducted after deployment, making remediation costly and time-consuming. On the other hand, back to the early phases of the Software Development Life Cycle (SDLC), software architects often lack the security expertise and feedback mechanisms needed to make informed design decisions, leading to vulnerabilities that remain undetected until later stages. In this paper, to address these challenges, we propose a research plan to integrate security considerations into the design phase. Our approach involves generating architecture-based penetration test cases, evaluating the attack surface of alternative architectures by using the generated test cases, and supporting software architects in making secure design decisions afterward. We plan to leverage threat modeling and Large Language Models (LLMs) to fulfill our target ambitions. To validate the applicability and effectiveness of our approach, we aim to conduct case studies in areas such as autonomous vehicles (AVs), which present significant security challenges.**

***Index Terms*—security by design, penetration testing, attack surface analysis, architectural decision support, autonomous vehicle systems**

I. Introduction

Nowadays, the integration of advanced software systems across various domains has made security a more challenging issue. According to general trends from Statista regarding cybercrime, the global cost of cybercrime was expected to exceed $8 trillion USD in 2023, with projections indicating it could rise to over $10.5 trillion by 2025 [1], which highlights the importance of security in today's digital world.

Penetration testing, also called pen testing, is a type of security testing defined as a proactive offensive technique for identifying, assessing, and mitigating security vulnerabilities [2]. In this process, the penetration testers act like attackers but with positive intentions. They, by mimicking the real attackers' behaviors, aim to penetrate the system to discover its security weaknesses for the purpose of mitigation. Penetration testing, which is typically run after the deployment phase, is essential for developing secure software systems. Still, it has some significant limitations that are mainly caused by the phase that is generally performed in the Software Development Life Cycle (SDLC). One issue is that security breaches are often discovered late in the development lifecycle when it's expensive and difficult to fix them [3]. Additionally, penetration testing relies heavily on the experience and skill of the testers, as they need to think like attackers and use creative approaches to uncover vulnerabilities [3], [4]. This makes the effectiveness of penetration testing directly tied to the tester's expertise. Other challenges include being time-consuming, requiring specialized skills, focusing mainly on known attacks, and being limited by budget constraints [5], which can all reduce its effectiveness and scope.

We argue that most of such limitations arise due to design flaws and poor architectural decisions, which lead to vulnerabilities that are often only discovered during later stages of development through penetration testing, causing a wide part of the limitations mentioned above. By this time, fixing them can be extremely costly and require considerable effort, as they may require substantial architectural changes. With this argument in mind, we are conducting current research aimed at shifting some penetration testing endeavors toward earlier in the design phase to promote a more secure architecture.

Our approach involves generating penetration test cases based on the software architecture as an input artifact. As a metamodel for the software architecture, we choose to use the Palladio Component Model (PCM) [6], a well-established ADL. However, our approach would not be restricted on only PCM and can be perfectly utilized on every architectural model that describe the structure and behavior of a software system. We are then willing to use these generated test cases to predict potential security breaches and evaluate the complexity of the attack surface of alternative architectural models. Additionally, we aim to assist software architects by providing actionable feedback for secure design decisions based on the analysis results derived from past steps, ultimately contributing to a more secure architecture in the end. To this end, we plan to utilize threat modeling and Large Language Models (LLMs).

The rest of the paper is organized as follows: In Section II, we will define the key terms used throughout the paper. In section Section III we will address the problems that motivate

This work was supported by funding from the topic Engineering Secure Systems of the Helmholtz Association (HGF) and by KASTEL Security Research Labs.

our research. In section Section IV we will model our research plan. First, we will formulate the target research questions, and then we will propose strategies for addressing them. In Sections Section V and Section VI, we will discuss the benefits of our research and review related works that have already been established, respectively. Finally, we will conclude our paper and evaluation strategy in Section Section VII.

II. Background

A. Penetration Testing

Penetration testing is a type of security testing aimed at improving the security of a system by identifying and mitigating its vulnerabilities [2]. According to the National Institute of Standards and Technology (NIST) definition, penetration testing involves a four-stage process: Planning, Discovery, Attack, and Reporting [7]. In the planning phase, no actual testing occurs; instead, boundaries and conditions for the test, such as relevant components and the scope of invasiveness is defined. The discovery phase involves systematically identifying all accessible external interfaces of the system, which form the initial attack surface, followed by vulnerability analysis to determine applicable vulnerability types. In the attack phase, testers actively attempt to exploit identified vulnerabilities to gain further access and uncover additional system components. Finally, the reporting phase occurs concurrently with the other phases, documenting all findings and their severity.

B. Threat Modeling

Threat modeling [8] is a technique based on the idea the earlier you find problems, the easier it is to fix them. Threat modeling is all about finding problems, and therefore, it should be done early in your development or design process or in preparing to roll out an operational system.

There are many ways to threat model. Asset-centric threat modeling is the approach that focuses on assets to find threats and model them accordingly. In this approach, assets are the valuable things you have, such as the things attackers want and the things you want to protect. Attacker-centric threat modeling is an approach that focuses on understanding the motivations, capabilities, and potential actions of attackers when analyzing threats to a system. Lastly, Software-Centric models are models that focus on the software being built or a system being deployed. So far, several promising methods for thread modeling have been introduced, considering the previously mentioned generic classification. STRIDE [9], LINDDUN [10], Attack Trees, PASTA, and Persona are some of the well-known methods, among others [11].

C. Large Language Models

Large Language Models (LLMs) are computational models that have the capability to understand and generate human language [12]. Typically, LLMs refer to Transformer language models that contain hundreds of billions (or more) of parameters, which are trained on massive text data, such as GPT-3, PaLM, Galactica, and LLaMA. LLMs exhibit strong capacities to understand natural language and solve complex tasks [14].

LLMs generally come with three distinct architectures: encoder-only models, encoder-decoder models, and decoder-only models [18]. Encoder-only models are primarily used for tasks that require understanding code, such as bug localization and vulnerability detection. In contrast, encoder-decoder models are utilized for tasks like code summarization which involve both understanding and generating code. Decoder-only models, such as Codex [13], are ideal for tasks that require sequential generation, including code generation and test case generation. In this context, tuning and prompt engineering are often employed for optimizing model performance [14].

III. Problems Statement and Motivation

In general, The motivation for conducting this research arises from the question of when penetration testing occurs within the SDLC. As discussed, penetration testing is generally performed after the deployment phase, when the software is fully deployed. The consideration of the place where penetration testing occurs, along with the lack of security measures in the early phases, leads to a transfer of vulnerabilities from the design phase to the deployment phase, which raises several critical issues. In this regard, four key problems form the basis of our motivation, which are outlined as follows:

P1 **Late Uncovered Issue** Penetration testing is typically carried out after the deployment phase of the SDLC on a running system. Conducting penetration testing late in the lifecycle can reveal issues when time and budget constraints limit the options and make it too tough for remediation. In reality, fixing problems at this stage is often prohibitively expensive [2]. The situation could become even worse when we consider that the breaches discovered during penetration testing are related to weaknesses in the software architecture. For example, the OWASP Top 10 highlights Insecure Design (A04:2021) [15] as a critical risk, emphasizing that a lack of secure architectural design decisions can lead to exploitable vulnerabilities in software systems, ultimately resulting in breaches. In such cases, the treatment process must address architectural issues by modifying software architecture and design choices to make them more secure. This means that the development process may need to go back several steps to the software architecture design phase, which can be extremely costly. This complex process doesn't end there. The new design decisions could also impact other parts of the system that have already been designed, implemented, and tested.

P2 **Reliance on Tester Experience, Expertise, and Domain Knowledge** Penetration testing is highly experience-based, as it requires testers to simulate real-world attacks by leveraging their expertise in security mechanisms, vulnerabilities, and potential exploits. Testers must think like attackers, using creative approaches that depend on their prior knowledge, intuition, and domain-specific expertise to uncover complex vulnerabilities that automated tools might miss. This reliance on both human expertise and domain knowledge poses a challenge, as the effectiveness of penetration testing is directly tied to the tester's skill, experience, and familiarity with the target system's context. [3], [4].

P3 **Time and Cost Consuming Process** Penetration testing is not only a time-intensive process but also a costly one [5]. Pen testing is generally divided into two main steps, both contributing to its high cost. The first step involves identifying existing vulnerabilities that are exploitable, a process that demands substantial time and resources. A key reason for this is the derivation and design of attack scenarios, where testers must creatively simulate potential attack vectors tailored to the system's architecture. This manual process requires deep expertise and can be time-consuming, adding to both the time and cost of the testing. The second step is focused on offering and implementing security measures to protect the system from these vulnerabilities, which adds further costs.

P4 **Lack of Design Feedback for Software Architects** The lack of feedback for software architects is a critical issue because their design decisions significantly impact the security of a system in the end. Design decisions, such as selecting authentication methods or security protocols, can either strengthen or weaken the system's defenses. However, architects often make these decisions in isolation, without receiving any feedback or early-stage security input. This absence of feedback means that poor design choices might go unaddressed, potentially introducing vulnerabilities that are only discovered later during costly and time-consuming penetration testing. Providing early feedback is essential to help architects make informed, secure design choices from the start, reducing vulnerabilities and improving overall security.

Upon reviewing the list, we conclude that issues 2 and 3 arise from the initial stages of penetration testing. During these early steps, penetration testers must adopt an attacker's mindset to identify valuable components of the system and create attack scenarios for identifying vulnerabilities. It is obvious that generating these attack scenarios in the early phases of penetration testing requires significant expertise from the testers, which can also make the process both time-consuming and costly. On the other hand, problems 1 and 4 are also relevant and arise from a common source. Both issues are due to a lack of security considerations in the early phases of the SDLC. This oversight leads to the transfer of vulnerabilities from the design phase to the later phases, where they are lately discovered during penetration testing.

IV. Research Plan

In the previous section, we described the problems that motivate our current research. In this section, we state our research questions, followed by the initial ideas we have for addressing them.

A. Research Questions (RQs)

We express our research concern through three distinct research questions (RQs) as follows:

RQ1 **How can architectural penetration test cases be generated based on architectural models during the design phase?** As briefly discussed previously, one reason that makes the penetration testing process complicated, highly time-consuming, and costly, and also dependent on the tester's experience and expertise, is the initial phase of penetration testing: designing appropriate attack scenarios. Through the first research question, our goal is to simplify this process by creating penetration test cases based on the architecture in the design phase. This will not only assist pen testers with pre-designed test cases but also serve as a starting point for creating secure software architecture in our subsequent research. Apart from that, prior to the mentioned main RQ, we aim to point out the question *'Do software architectural models provide good enough information for penetration test case generation?'* as a side research question in this part.

RQ2 **How can the generated test cases be prioritized to facilitate penetration testing efforts?** During penetration testing, prioritizing test cases helps security professionals focus on the most critical vulnerabilities, especially when time and resources are limited or when the target system is complex with numerous test cases. This process not only improves the effectiveness of penetration testing by identifying the most urgent threats, but it also can be used to provide valuable insights for software architects when designing secure architecture. The ranked test cases can serve as complementary leverage to provide precise feedback, helping architects make informed design decisions to enhance the system's security.

RQ3 **How can generated test cases be used to identify security risks and rate architectural models based on their security levels?** Software architects typically design the architecture by making a set of design decisions, which ultimately affects the overall software quality, especially security. In this research question, we aim to provide software architects with recommendations on the design decisions made, explicitly focusing on security. This will help the development process by addressing security issues at the architectural level, preventing them from being overlooked until later phases. This goal will also address the issue of late uncovered security breaches by finding and mitigating them in the early design phase.

Figure 1 shows how target research questions are integrated into the standard Software Development Life Cycle. While the demonstration adheres to the typical software development process, the approach is generally fed by architectural models. Consequently, it can be effectively applied to various methodologies where software architecture plays a role.

B. Preliminary Strategies

RQ1. How can architectural penetration test cases be generated based on architectural models during the design phase?

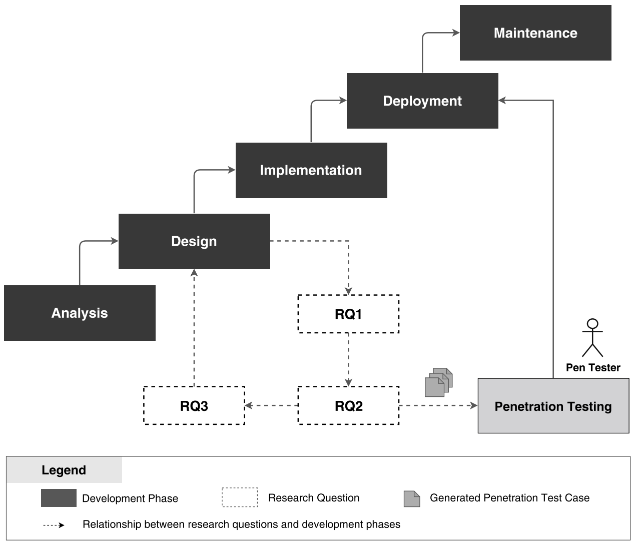

Fig. 1. Integration of Target Research Questions into the SDLC.

To generate penetration test cases, the first crucial step is to identify the available resources and analyze them to create potential attack scenarios. This initial phase, known as the information-gathering step, involves collecting as much information as possible about the system being tested. The information-gathering phase can be approached from two different perspectives: The white-box approach, involves collecting and analyzing existing development artifacts such as software architecture, and the Black-box approach, which entails gathering information from public sources such as product, technical, or functional descriptions using tools like port scanning, recording, and sending bus messages [16].

In our case, we will stay with the white-box approach to analyze the software architecture to achieve the goal. Hence, to address RQ1, we will employ three different strategies, as follows, each increasing in complexity, and we expect to increase productivity as well.

Strategy 1. Threat Modeling-based Strategy We aim to use threat modeling techniques to generate penetration test cases through this strategy. Jürgen et al. [17] highlight the advantages of employing threat modeling in the context of automotive cybersecurity. Their findings indicate that threat modeling can be effectively employed to generate threat scenarios, attack scenarios, conduct risk assessments, and serve other valuable purposes in this field. The idea involves two diverse phases: (1) Performing Threat modeling based on the architectural models, (2) Utilizing thread modeling artifacts, like attack trees, to generate penetration test cases.

Besides, as one further interesting sub-question, we intend to explore which threat modeling technique(s) could improve the generation of penetration test cases from architectures.

Strategy 2. LLMs-Empowered Strategy Recently, LLMs have greatly contributed to improving various cybersecurity tasks, such as vulnerability detection and penetration testing [19], [20]. Through this strategy, we plan to incorporate LLMs into the early phases of the SDLC to generate architectural penetration test cases. To the best of our knowledge, this approach will be the first to utilize LLM technology specifically for the task of generating penetration test cases from architectural models. Consequently, beforehand, the question *'Can LLMs contribute beneficially to the generation of architectural penetration test cases?'* should be examined. This is another research line we would like to focus during this strategy.

To achieve this, we propose to initially tackle the problem with an algorithm that does not involve LLMs. The goal is to generate test cases based on input architectural models using collections such as the Common Weakness Enumeration (CWE) [21], the Common Attack Pattern Enumeration and Classification (CAPEC) [22], and the Common Vulnerabilities and Exposures (CVE) [23]. CWE is a community-developed list of common software and hardware weaknesses that can lead to vulnerabilities, CAPEC is a comprehensive dictionary of known attack patterns that describe the techniques attackers use to exploit vulnerabilities and CVE is a list of publicly disclosed cybersecurity vulnerabilities.

Once we developed the algorithm and confirmed its positive results — indicating that generating penetration test cases from architectural models using these resource collections is feasible — we will proceed to analyze how we can utilize LLMs to enhance the generation process. In this context, LLMs could either completely replace certain aspects of the algorithm or be used to improve specific parts of it. This integration is anticipated to make our approach more scalable, efficient, and accurate, while also expanding the range of test coverage. Furthermore, the developed algorithm could serve as a valuable comparison baseline to evaluate the new LLM-empowered approach and demonstrate its beneficial contributions.

Strategy 3. Hybrid LLMs-Empowered Strategy Tuned with Threat Modeling Artifacts Since collections like CWE and CAPEC document security weaknesses and attack paths across the general cybersecurity domain, independent of specific domains of concern, incorporating LLMs into the pen test case generation process using in a way that proposed in the second strategy, may keep the model relatively general that leads us to receive broad responses from the model, irrespective of the particular domain of concern, the automotive domain. Thus, we expect the model to produce penetration test cases that are relevant to the overall cybersecurity context through the second strategy.

Given this limitation, we propose a third strategy focused on narrowing the LLMs' focus down to the domain of autonomous vehicles. In simpler terms, our main idea is to develop a hybrid solution based on LLMs that are tuned using threat models specific to autonomous vehicles and guided by prompt engineering techniques. This specialized approach will help the model concentrate on our target domain of concern, produce more accurate penetration test cases with better coverage, and ultimately enhance its overall performance.

RQ2. How can the generated test cases be prioritized to facilitate penetration testing efforts?

The definition of penetration testing highlights the clear relationship between penetration test cases and vulnerabilities. The primary goal of the penetration testing process is to uncover existing vulnerabilities through the use of these test cases. By understanding this relationship, we can prioritize the test cases based on the vulnerabilities they address.

Several factors can influence the prioritization of a vulnerability and its corresponding penetration test case. These factors may include severity, likelihood of occurrence, remediation cost, and other relevant contextual elements. However, to the best of our knowledge, there is currently no formal method to systematically calculate these factors at the model level. To address this gap, we propose using historical data to establish appropriate values for these attributes.

In summary, to answer our research question, we aim to develop an algorithm that can automatically prioritize penetration test cases based on their severity, using historical data. For our preliminary data, we suggest utilizing the National Vulnerability Database (NVD), which extends the CVE database by providing additional metadata for each vulnerability, including CVSS scores (severity ratings) [24].

RQ3. How can generated test cases be used to identify security risks and rate architectural models based on their security levels?

The design of software architectures permanently raises an essential question for software architects: is the designed architecture secure? Are the design decisions made secure, or are there more secure alternatives available? However, many software architects often lack in-depth knowledge of security, making it challenging to address these concerns during the design phase, which can lead to poor, insecure architectures.

To address this issue, through this RQ, we plan to assist software architects by comparing and rating architecture alternatives based on their attack surface complexity. We will utilize generated and ranked penetration test cases derived from our previous research questions (RQ1 and RQ2) to identify the most security-sensitive aspects of the architectures. This process will allow us to assess the corresponding design decisions and evaluate the attack surfaces. Our goal is to provide valuable insights in the form of a recommender system to assist software architects in making more informed decisions regarding security.

V. Benefits

From both industrial and academic perspectives, the study addresses a critical gap in the software development lifecycle by empowering software architects with practical security skills and insights. Architects often lack deep security expertise, making it challenging to make informed design decisions that proactively mitigate vulnerabilities, resulting in more secure software architecture. This approach supports architects by providing actionable security recommendations during the early architectural design stages, enabling them to embed security into their designs from the outset. Moreover, the approach offers significant value to penetration testers by generating preliminary penetration test cases derived from the architectural model, the most crucial layer of a software system. In addition to these benefits, this method brings security considerations into the early phases of the software development lifecycle, contributing to a ”security by design” philosophy and bridging the gap between penetration testing and design phase.

VI. Related Work

Security by design, the concept that refers to integrating efforts and security measures in the early stages of software development to identify and mitigate security incidents early on, making the software more reliable at a lower cost and with less effort, has always been a topic of interest among both industry and academics in the software engineering direction.

Tobias [25] conducted a doctoral proposal on a somewhat relevant direction within the CPS domain to target the experience-based and no-feedback loop problem of penetration testing and software architecture design by modeling attacker behavior to tackle them. Ralf et al. [26] developed a metamodel for analyzing attackers and system vulnerabilities to generate attack paths more effectively. They created models for both attackers and vulnerabilities to analyze and create attack paths based on the system’s architectural models. These models are established based on well-known vulnerability classifications such as CWE, CVE, and CVSS. Another study by the same authors [27] suggests a method for generating architectural attack paths through the construction and traversal of attack graphs. In comparison to their earlier study, the attack graph now includes all vulnerabilities unless they possess specific filtered properties. Haralambos et al. [28] introduce an attack graph-based approach for generating attack paths and predicting cyber-attacks using recommender systems. The primary objective of this research is to identify attack paths and demonstrate how a recommendation method can be utilized to classify future cyber-attacks for risk management purposes. In addition to those studies mentioned above, several other research has been conducted to leverage asset-centric threat modeling techniques to resolve the same problem. In [29], the authors focused on the automotive domain and introduced a new approach for identifying attack scenarios. They utilized the STRIDE (spoofing, data tampering, repudiation, information disclosure, denial of service, and elevation of privilege) threat modeling technique and carried out Attack Tree Analysis (ATA). This tree-based approach aims to identify potential attack strategies that an attacker may use against the system under test. The study [30] has been conducted to address the experience-based drawbacks that uses knowledge of existing security attacks on vehicles to automate the security testing process. Lastly, Jürgen et al. [16] present an approach for generating penetration test cases by leveraging Hazard

Analysis and Risk Assessment and attack trees as a threat modeling technique.

Despite the outstanding contributions already made in this area such as attack path generation, attacker capabilities modeling, and vulnerability modeling, there still remains a lack of a design-level feedback loop that could benefit from these advancements to assist software architects in making secure design decisions which is our ambitious focus part in our planned research. Apart from that, supporting penetration testers with some preliminary pen test cases generated from architectures is another line we would like to achieve.

VII. Conclusion

This paper presented our idea on integrating security considerations into the early phases of the SDLC. We identified three major problems associated with typical penetration testing which mostly is conducted late in the process, as well as one design level issue that leads to the discovery of vulnerabilities in later phases, resulting in remediation being both timely and costly. These problems motivated us to emphasize the importance of incorporating security measures during the initial steps of the SDLC.

To address this, we introduced our plan to generate architecture-based penetration test cases, prioritize them based on their severity, and then utilize them to evaluate the complexity of attack surfaces in different architectural alternatives, assisting architects toward more secure architecture design accordingly. In this light, we formulated three distinct research questions and proposed our preliminary ideas to tackle them. Lastly, we demonstrated the significant benefits that this study would have from both industrial and academic points of view.

In terms of evaluation, we aim to adopt a case study-based approach that involves test case generation, prioritization, and feedback mechanisms in the context of autonomous vehicles. We will measure key performance indicators such as accuracy, precision, recall, and overall effectiveness in identifying and mitigating vulnerabilities. Additionally, we will create a comparison between our proposed approach and similar existing methods by benchmarking the results from the case study.

References

[1] Statista; Statista Technology Market Insights, “Estimated cost of cybercrime worldwide 2018-2029.” Accessed: Sep. 09, 2024. [Online]. Available: https://www.statista.com/forecasts/1280009/cost-cybercrime-worldwide.

[2] B. Arkin, S. Stender, and G. McGraw, “Software penetration testing,” IEEE Secur. Privacy Mag., vol. 3, no. 1, pp. 84–87, Jan. 2005, doi: 10.1109/MSP.2005.23.

[3] M. Felderer, P. Zech, R. Breu, M. Büchler, and A. Pretschner, “Model-based security testing: a taxonomy and systematic classification: MODEL-BASED SECURITY TESTING,” Softw. Test. Verif. Reliab., vol. 26, no. 2, pp. 119–148, Mar. 2016, doi: 10.1002/stvr.1580.

[4] F. Luo et al., “Cybersecurity Testing for Automotive Domain: A Survey,” Sensors, vol. 22, no. 23, p. 9211, Nov. 2022, doi: 10.3390/s22239211. Verif. Reliab., vol. 26, no. 2, pp. 119–148, Mar. 2016, doi: 10.1002/stvr.1580.

[5] J.-P. A. Yaacoub, H. N. Noura, O. Salman, and A. Chehab, “A Survey on Ethical Hacking: Issues and Challenges,” 2021, arXiv. doi: 10.48550/ARXIV.2103.15072.

[6] R. Reussner et al., Eds., Modeling and simulating software architectures: the Palladio approach. Cambridge, Massachusetts London, England: The MIT Press, 2016.

[7] K. A. Scarfone, M. P. Souppaya, A. Cody, and A. D. Orebaugh, “Technical guide to information security testing and assessment.,” National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 800-115, 2008. doi: 10.6028/NIST.SP.800-115.

[8] A. Shostack, Threat modeling: designing for security. Indianapolis, IN: John Wiley and Sons, 2014.

[9] R. Khan, K. McLaughlin, D. Laverty, and S. Sezer, “STRIDE-based threat modeling for cyber-physical systems,” in 2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe), Torino: IEEE, Sep. 2017, pp. 1–6.

[10] “linddun.org — Privacy Engineering.” Accessed: Dec. 11, 2024. [Online]. Available: https://linddun.org/.

[11] N. Shevchenko, T. A. Chick, P. O’Riordan, T. P. Scanlon, and C. Woody, “Threat modeling: a summary of available methods,” Software Engineering Institute— Carnegie Mellon University, pp. 1–24, 2018.

[12] Y. Chang et al., “A Survey on Evaluation of Large Language Models,” ACM Trans. Intell. Syst. Technol., vol. 15, no. 3, pp. 1–45, Jun. 2024, doi: 10.1145/3641289.

[13] “OpenAI Codex.” Accessed: Jan. 24, 2025. [Online]. Available: https://openai.com/index/openai-codex/.

[14] W. X. Zhao et al., “A Survey of Large Language Models,” Nov. 24, 2023, arXiv: arXiv:2303.18223.

[15] “A04 Insecure Design - OWASP Top 10:2021.” Accessed: Jan. 23, 2025. [Online]. Available: https://owasp.org/Top10/A04_2021-Insecure_Design/.

[16] J. Dürrwang, J. Braun, M. Rumez, R. Kriesten, and A. Pretschner, “Enhancement of Automotive Penetration Testing with Threat Analyses Results,” SAE Int. J. Cybersecurity, vol. 1, no. 2, pp. 91–112, Nov. 2018,

[17] J. Dobaj, G. Macher, D. Ekert, A. Riel, and R. Messnarz, “Towards a security-driven automotive development lifecycle,” J Software Evolu Process, vol. 35, no. 8, p. e2407, Aug. 2023, doi: 10.1002/smr.2407.

[18] X. Hou et al., “Large Language Models for Software Engineering: A Systematic Literature Review,” 2023, arXiv. doi: 10.48550/ARXIV.2308.10620.

[19] A. Ding, G. Li, X. Yi, X. Lin, J. Li, and C. Zhang, “Generative Artificial Intelligence for Software Security Analysis: Fundamentals, Applications, and Challenges,” IEEE Softw., pp. 1–8, 2024, doi: 10.1109/MS.2024.3416036.

[20] H. Xu et al., “Large Language Models for Cyber Security: A Systematic Literature Review,” 2024, arXiv. doi: 10.48550/ARXIV.2405.04760.

[21] “Common Weakness Enumeration (CWE).” Accessed: Sep. 15, 2024. [Online]. Available: https://cwe.mitre.org/.

[22] “Common Attack Pattern Enumeration and Classification (CAPEC).” Accessed: Sep. 15, 2024. [Online]. Available: https://capec.mitre.org/.

[23] “Common Vulnerabilities and Exposures (CVE).” Accessed: Sep. 15, 2024. [Online]. Available: https://cve.mitre.org/.

[24] “National Vulnerability Database (NVD).” Accessed: Sep. 15, 2024. [Online]. Available: https://nvd.nist.gov/.

[25] T. Walter, “Architectural Pen-Test Generation and Vulnerability Prediction for Cyber-Physical Systems,” in 2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C), Honolulu, HI, USA: IEEE, Mar. 2022, pp. 45–46. doi: 10.1109/icsa-c54293.2022.00016.

[26] M. Walter, R. Heinrich, and R. Reussner, “Architectural Attack Propagation Analysis for Identifying Confidentiality Issues,” in 2022 IEEE 19th International Conference on Software Architecture (ICSA), Honolulu, HI, USA: IEEE, Mar. 2022. doi: 10.1109/icsa53651.2022.00009.

[27] M. Walter, R. Heinrich, and R. Reussner, “Architecture-Based Attack Path Analysis for Identifying Potential Security Incidents,” in Software Architecture, B. Tekinerdogan, C. Trubiani, C. Tibermacine, P. Scandurra, and C. E. Cuesta, Eds., Cham: Springer Nature Switzerland, 2023.

[28] N. Polatidis, E. Pimenidis, M. Pavlidis, S. Papastergiou, and H. Mouratidis, “From product recommendation to cyber-attack prediction: generating attack graphs and predicting future attacks,” Evolving Systems, vol. 11, no. 3, pp. 479–490, Sep. 2020, doi: 10.1007/s12530-018-9234-z.

[29] Z. Abuabed, A. Alsadeh, and A. Taweel, “STRIDE threat model-based framework for assessing the vulnerabilities of modern vehicles,” Computers & Security, vol. 133, p. 103391, Oct. 2023, doi: 10.1016/j.cose.2023.103391.

[30] F. Sommer and R. Kriesten, “Attack Path Generation Based on Attack and Penetration Testing Knowledge,” Jan. 2023.