Journal of Software: Evolution and Process

SPECIAL ISSUE - EMPIRICAL PAPER OPEN ACCESS

Empowering Software Engineers to Design More Secure Web Applications: Guidelines and Potential of Using LLMs as a Recommender Tool

Raffaela Groner[1] | Klara Svensson[1] | Drake Axelrod[1] | Ranim Khojah[1] | Mazen Mohamad[1,2] | Rebekka Wohlrab[1]

[1]Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden | [2]Dependable Transport Systems, Research Institutes of Sweden (RISE), Gothenburg, Sweden

Correspondence: Raffaela Groner (raffaela@chalmers.se)

Received: 5 June 2025 | **Revised:** 14 October 2025 | **Accepted:** 2 January 2026

Keywords: cybersecurity | design science research | experiments | guidelines | interviews | LLM | software engineering | survey | web applications

ABSTRACT

As software applications get increasingly connected and complex, cybersecurity becomes more and more important to consider during development and evaluation. Software engineers need to be aware of various security threats and the countermeasures that can be taken to mitigate them. Currently, there is a lack of guidance for software engineers aiming to develop secure web applications. We conducted a design science research study, resulting in a set of guidelines to aid software engineers in developing secure web applications. The set of guidelines was constructed based on interview data with 10 industry practitioners. These guidelines were then evaluated using a survey with 28 respondents. Additionally, we conducted experiments in which we provided a large language model with our guidelines and vulnerability reports as input. The large language model should extend the given vulnerability reports by recommending which of our guidelines can help prevent the given vulnerability in the future. The extended reports were evaluated by two external researchers experienced in cyber security and one author. Our results indicate that developers consider using these proposed guidelines for the development and assessment of secure web applications in different stages of the software development lifecycle. Our results also show that it is possible to automatically enhance vulnerability reports to support developers meaningfully and that the guidelines recommended by the large language model are useful to prevent the respective vulnerabilities in the future.

1 | Introduction

As the dependence on software applications in various sectors grows, and software systems become more complex, it is becoming increasingly important for software engineers to make sure that the software they create and maintain is secure [1]. Previous research [1] found that developers lack proper security knowledge. While software engineers are responsible for designing and developing software, cybersecurity professionals are typically responsible for mitigating security vulnerabilities. To proactively mitigate vulnerabilities, software engineers need to be familiar with various security threats and with the mitigation strategies that can be applied to reduce the risk of these threats [2]. Web development is common among developers with a wide range of expertise due to the low threshold of web application development. However, the risk of exploiting sensitive data within web applications is high due to their easy public accessibility. Thus, it is even more important that cybersecurity is considered [3, 4].

The intersection between cybersecurity and software engineering has gained increased attention in the last years [1, 2, 5–8]. They mostly focus on tools that can be used during development, such as recommender systems for secure coding [9] or mechanisms to create more secure software supply chains [10]. Previous studies often do not focus specifically on guidelines and resources for developers. Education and training for developers are needed to convey an understanding and awareness of cybersecurity concerns. Those concerns are not only relevant during the implementation phase, but also during other lifecycle phases, from planning to testing and maintenance. This research aims to fill this gap and identify guidelines for meeting the information needs to develop secure web applications. To this end, we examine the information needs of software engineers and the current role of cybersecurity in developing secure web applications. Additionally, we examine how vulnerability scanners can be enhanced with guidelines recommended by a large language model (LLM) to make the relevant guidelines easily accessible for developers and help them prevent similar vulnerabilities in the future.

We present the results of a design science research study [11] consisting of cycles iterations. This article presents our results from our third cycle, and is an extension of our work presented in [12]. In our previous work [12], we developed a set of guidelines to aid software engineers in developing more secure web applications using two cycles. These guidelines are targeted towards software engineers as they play an important role in ensuring secure systems through incorporating security when coding and implementing best practices during the development lifecycle [5]. We also identify the software life-cycle phase that these guidelines can be integrated into. Additionally, we collected empirical evidence from professionals within the fields of software engineering, cybersecurity, and management. Through cooperation with an external partner,[1] the study gained input from industry practitioners and feedback on the usefulness of the created guidelines. In our previous work, we focused on the following contributions:

1. An analysis of the information needs of developers aiming to implement secure web applications;
2. A set of guidelines to develop secure web applications;
3. An assessment of the perceived usefulness of the guidelines, based on a survey.

In this work, we extend our previous work [12] with a third cycle in which we examine the practical integration of our guidelines into vulnerability reports. We analyzed five well-known PHP web applications with the help of the security scanner Aikido. Although new web applications are implemented with the help of JavaScript or Python, 74% of the web applications whose server implementation is known are mainly implemented in PHP [13]. Thus, maintaining and vulnerability management for PHP is still extremely relevant in practice even though new web technologies are on the rise.

In our experiments, we analyzed whether we could extend the vulnerability reports we received with the help of a LLM to make our guidelines more accessible to developers, link them specifically to vulnerabilities, and support developers in preventing these vulnerabilities in the future. The resulting extended vulnerability reports were independently evaluated by two external researchers experienced in cyber security and one author with experience in security modeling. Our results show that an LLM can meaningfully enhance vulnerability reports by our guidelines. Additionally, all evaluators assessed the guidelines connected to a vulnerability by the LLM as useful to prevent the respective vulnerability in the future. Therefore, we extend our previous work [12] with the following contributions:

1. A more in-depth analysis and extended findings on the assessment of our guidelines from our previous survey.
2. An LLM-based approach that recommends our guidelines to support developers in preventing vulnerabilities in the future.
3. An evaluation of the LLM-based approach with experts in security.

In the following, we discuss related work and we provide background on the security scanner, we used to collect data for our experiments. Subsequently, we describe our research method in Section 3 and present our findings in Section 4. We discuss our results in Section 5 and conclude in Section 6.

2 | Related Work

In this section, we discuss related work. In Section 2.1, we focus on available resources for developing secure web applications. Subsequently, we discuss related work in the area of secure coding with the help of LLMs in Section 2.2. Finally, we introduce the static security analysis and the static application security testing (SAST) functionality that we use in our study.

2.1 | Resources for Developing Secure Web Applications

Several resources for developers aiming to create secure web applications exist. For instance, the Mitre Att&ck (MITRE) framework [14] provides comprehensive adversary tactics and techniques data. Open Web Application Security Project (OWASP)[2] is a repository of resources for developers to make web applications more secure. OWASP is the gold standard for security-related information in the industry. However, OWASP mostly includes the Top 10 vulnerabilities and no concrete steps on how these vulnerabilities can be mitigated using threat modeling and other techniques. Our work partially relies on these resources and aims to make them more accessible to developers. Recently, Siderova et al. [15] studied approaches to integrating security concerns into model-driven engineering (MDE) [16] for web applications. They concluded that approaches are needed that better address the OWASP Top 10 vulnerabilities. Additionally, they concluded that researchers should conduct more empirical studies on the topic. We focus on these issues in this article.

A systematic literature review on developer-centric development [8] concluded that learning support in the reported literature takes the form of IDE plugins with limited practical validation. The authors also discussed the lack of links to external resources for developers' training, which we target in our study. The authors also discuss that the majority of reviewed papers suggest providing feedback to developers actively, meaning that it requires interrupting them while conducting their work, which might be an issue. In our study, we take a passive approach by providing guidelines for the developers to learn from, that is, support on request. Lastly, the literature review discussed the difference between practitioners and students as research participants. While the majority of the studies feature students, we have a mix of both, which gives us diverse feedback.

Weir et al. performed workshop-based interventions in eight organizations combined with interviews to assess the impact of them. It was found that development teams, when suitably guided, for example, by workshops in which possible threats and their impact are discussed, can significantly improve their security maturity even without the support of security specialists [17]. Another study [18] analyzed general advice resources and described the need for evaluating these resources and developing new ones.

Assal and Chiasson [2] examined the human factors of software security and argued that a better understanding of human behavior and motivation can lead to more effective security policies and training methods. Similarly, the importance of integrating security policies and strategies into the development cycle was highlighted in a study on best practices for ensuring security in DevOps [19]. Our research builds upon these studies by conducting research from a software engineering perspective in web application development, as well as introducing valuable empirical evidence from industry practice.

Researchers have also looked into the challenges of implementing DevSecOps practices [20]. They emphasized the need for security standards for code, security tools (e.g., for testing), and security training and education for engineers. Our article focuses on creating such guidelines that developers can follow to create more secure web applications.

Several other studies have proposed models for integrating security into the Software Development Life Cycle (SDLC). One such model is the Secure Software Development Model [6], which is based on international standards and best practices. This model covers the entire development process without a specific focus on developers. As a result, there is a lack of detailed guidelines that specifically support developers. Kalhoro et al. [7] proposed a cyber hygiene model that identifies key factors influencing software engineers' cyber hygiene behavior. The model is a valuable asset when developing guidelines for secure web application development.

Gasiba et al. [1] addressed the need for effective training methods using a game to raise awareness and provide practical advice. Finally, the study by Tahaei and Vaniea [8] examined developer-centered security and provided tools and processes to better support both developers and secure code production. However, this study did not focus specifically on web applications and suggested further study into developer-focused security. Our research aims to further explore developer-centered security as related to web applications, by developing and evaluating guidelines for different lifecycle phases.

2.2 | Development of Secure Source Code using LLMs

LLMs have lately been studied to assess their applicability and usability in various cybersecurity aspects. In particular, researchers have been using LLMs to detect vulnerabilities in web-development code [21], identify weaknesses and attacks [22–24], and propose mitigation strategies to different cybersecurity threats [25].

Other applications utilize the language generation abilities of LLMs and investigated their potential in providing textual feedback and security-related tips for developers. Liu et al. [26] proposes a framework where they rely on natural language communication between the developer and the LLM for secure coding. Their framework that utilizes reward-based prompt engineering improved the vulnerability understanding of the LLM by rewarding the LLM for each vulnerability found and successfully fixed. Yosifova [27] uses an LLM as an assistant for security best practices in IoT [27]. The study presents an experiment in which a user prompts an LLM asking about different secure coding practices and finds that the output generated by the LLM aligns with the OWASP guidelines for secure coding. Sajadi et al. [28] study the ability of LLMs to promote secure coding and security awareness. The study uses Stack Overflow vulnerability-related questions to prompt the LLMs and investigates whether the LLM are capable of providing answers that contribute towards raising the developers' awareness of the security issues. The findings of Sajadi et al. [28] show that while the LLMs perform poorly in detecting the vulnerabilities, they do provide better information regarding the cause and fixes to the detected ones compared to the answers provided by Stack Overflow users.

In this study, we focus on examining if LLMs can take an expert role in guiding the developers on the cause of vulnerabilities while suggesting guidelines to follow in order to avoid similar vulnerabilities. In contrast to related studies, we use a predefined set of guidelines as a context for the prompt to the LLM and ask for specific clauses for the output. This helps us evaluate whether the LLM is able to give more specific and relevant recommendation from established guidelines.

2.3 | Background on SAST

We used the SAST functionality provided by Aikido to identify vulnerabilities. The SAST engine of Aikido follows a rule-based approach where for each vulnerability to be checked, a rule is defined [29, 30]. More precisely, the analysis is based on semgrep rules [31] that contain, among other things, a code pattern that describes the vulnerable code. A simplified example of such a rule can be seen in Listing 1. This rule should trigger a warning (`severity`) when a value is assigned to innerHTML (`pattern-either`) in a JavaScript program (`languages`).

```
rules:
  - id: assign-to-innerHTML
    message: Using innerHTML to insert text into a web page can be a security risk.
    languages:
      - javascript
    severity: WARNING
    pattern-either:
      - pattern: $ELEMENT.innerHTML += $SOMETEXT
      - pattern: $ELEMENT.innerHTML = $SOMETEXT
```

LISTING 1 | Simplified semgrep rule to check for assignments to inner HTML.

Aikido offers 226 rules to check for vulnerabilities in 16 languages including PHP, JavaScript/TypeScript, and GitHub Actions [29]. Based on these rules, the SAST functionality generates a vulnerability report that contains an entry for each code snippet that matches a rule. Such an entry describes, among other things, the affected lines of code, the corresponding file, and the name of the matched rule.

In our experiments, we consider a total of 24 vulnerability types that Aikido has identified for the projects under consideration. Table 1 provides a description of the 24 vulnerabilities.

3 | Research Method

The purpose of this study is to gain a better understanding of the current web application security knowledge within the software engineering domain, design a set of guidelines for developing secure web applications, and examine to what extent these guidelines can be leveraged to improve security.

Because we aim for a practical solution for developing secure web applications, we apply design science research as a research method. Design science involves iteratively designing, developing, and evaluating the artifact [11]. In our case, the artifact is a set of guidelines to aid software engineers in developing more secure web applications. Design science is an appropriate method for researchers who want to get an in-depth understanding of the problem and create a solution that is evaluated and refined in several cycles.

To understand the current state of the practice and investigate the potential use of guidelines, our research focuses on the following research questions:

RQ1. What are the security information needs of software engineers to develop secure web applications?
RQ2. What potential guidelines can support software engineers in developing secure web applications?
RQ3. How can those guidelines be applied to the development of secure web applications?
RQ4. To what extent can those guidelines be leveraged to improve the security of web applications with LLMs?

In order to answer our research questions, we performed three cycles within design science research as illustrated in the overview in Figure 1. In this figure, we have highlighted the research questions that are part of our previously published work [12] with blue circles. The research question RQ4 added in this extension is highlighted in green. In the following Sections 3.1 and 3.2, we describe the research methods applied in the first and second cycle, which are part of our previous work [12]. In Section 3.3, we present the research methods applied in the second and third cycles. Subsequently, we describe the data analysis performed for each cycle.

3.1 | First Cycle: Formulation of Guidelines

This cycle focused on learning more about what kinds of resources and information software developers require to develop secure web applications and how to provide these resources as a set of guidelines. This was done by reviewing the literature and conducting semistructured interviews.

3.1.1 | Interview Design

The interviews were designed to gather information on the topics of software engineering practices for designing secure web applications, the most common problems, and effective preventative practices. The interview guide is available on Figshare.[3] Note that we could not make the interview transcripts available because we assured the interviewees that their data would be treated confidentially. The interviews were conducted through a video conferencing system. All interviews were recorded and transcribed.

3.1.2 | Selection of Participants

We aimed to understand how software engineers can be supported in developing more secure web applications. To this end, we selected a mix of participants. All of them worked at the same company.[4] Some were cybersecurity experts, whereas others came from the web application domain. We decided to include both experienced and junior participants, given that the target audience of our guidelines is rather broad. The interviews were conducted with individuals of different roles with at least six months of experience in their current positions. Some of the interviewees had overlap among the domains which resulted in six interviewees from the software engineering domain, three from the management domain, and five from the cybersecurity domain. All interviewees are involved in activities connected to web application development.

Table 2 shows an overview of the interviewees in the first cycle.

The guidelines were created in the first cycle using the information gathered from the interviews and the literature, more

TABLE 1 | Overview of the 24 vulnerability types considered in our experiments.

Vulnerability type	Description
Prototype pollution vulnerability detected	The prototype mechanism of JavaScript is exploited, for example, to override functions [32].
Potential SQL injection via Laravel function	The PHP framework Laravel [33] includes a template engine that offers a command to display unescaped data, which can be exploited by Cross-Site Scripting (XSS) [34] attacks that inject malicious scripts into websites [35].
SSL certificate verification turned off during requests	The option CURLOPT_SSL_VERIFYPEER in curl is set to 0. Thus, the authenticity of the peer's SSL certificate is not checked [36].
Using potentially unsafe FTP connections to move data	The PHP function ftp_connect is used to open an insecure FTP connection [37].
HTTP request might enable SSRF attack	The code sends an HTTP request to a URL that is assembled using variables whose data source is unclear. If the variable values can be manipulated by users, the server might connect to a malicious third-party system [38].
Path traversal attack possible via file functions	The code uses functions like fopen [39] and provides the filepath in a variable whose data source is unclear. Thus, an attacker might get access to other files or directories [40].
Rendering unescaped input in HTML template can lead to XSS attacks	The template engine Twig for PHP allows rendering input from a variable in HTML [41]. If this input is not escaped, it can be exploited to perform XSS attacks [42].
Template Injection in GitHub Workflows Action	The yml file defining GitHub Workflows Action contains double curly braces whose input is evaluated by node.js and can therefore be exploited [43].
Rendering unescaped input can lead to XSS attacks	The PHP functionecho is used to render the value of a variable without checking the content of the variable [44].
Using phpinfo() can expose sensitive info to users	The PHP function phpinfo discloses a lot of system information, some of which is highly sensitive [45].
Using unsafe GitHub Actions trigger may allow privilege escalation via CI/CD	Trigger events like pull_request_target in a yml file can, for example, lead to granting attackers write permissions to a repository by checking out a malicious pull request [46].
A timing attack might allow hackers to brute-force passwords	The JavaScript/TypeScript code compares hashes to ensure integrity and authenticity. Node.js stops comparing values after the first mismatch. Thus, attackers can measure the duration necessary to compare values to obtain the compared values [47].
Using backticks in PHP can lead to remote code execution	PHP executes arbitrary expressions in backticks, which can be exploited for remote code execution [48].
NoSQL injection attack possible	The MongoDB function findOne uses an optional parameter that uses query operators to specify the selection criteria [49, 50]. The operator “where” uses JavaScript expressions to find documents and is therefore vulnerable to injection attacks [51].
Using var_dump() can expose sensitive info to users	The PHP function var_dump() returns detailed information about the data structure and values of a variable, disclosing sensitive information that can be exploited [52].

(Continues)

TABLE 1 | (Continued)

Vulnerability type	Description
Remote Code Execution possible via eval()-type functions	The JavaScript function eval executes any JavaScript code given as a parameter, enabling attackers to run arbitrary code [53].
Using v-html in Vue templates can lead to XSS attacks	The attribute v-html can be used to render HTML code in a Vue application. The value of the attribute is not escaped. Thus, this attribute can be exploited to perform an XSS attack [54].
Unsafe eval usage can lead to remote code execution	The PHP function eval executes any PHP code given as a parameter, enabling attackers to run arbitrary code [55].
Third-party GitHub Actions should be pinned	If a third-party GitHub Action is not pinned to a full-length commit SHA, there is a risk that an attacker could install a backdoor in the action repository [56].
Unsafe exec usage can lead to remote code execution	The PHP function exec executes an external program based on a given command, enabling attackers to run arbitrary programs [57].
Using unserialize can lead to remote code execution	The PHP function unserialize can be exploited to execute arbitrary code due to object instantiation and autoloading in PHP [58].
Using document write methods can lead to XSS attacks	In JavaScript, the DOM property innerHTML can be exploited to perform XSS attacks because its content is rendered without sanitization [59].
Potential SQL injection via string-based query concatenation	If a SQL query is constructed by appending strings and parameters without sanitization, the query can be influenced by an attacker, and the application will run malicious code on the database [60].
Rendering unescaped input in handlebar/mustache template can lead to XSS attacks	The template systems mustache [61] and handlebars [62] render given HTML expressions without sanitization, enabling attackers to perform XSS attacks [63].

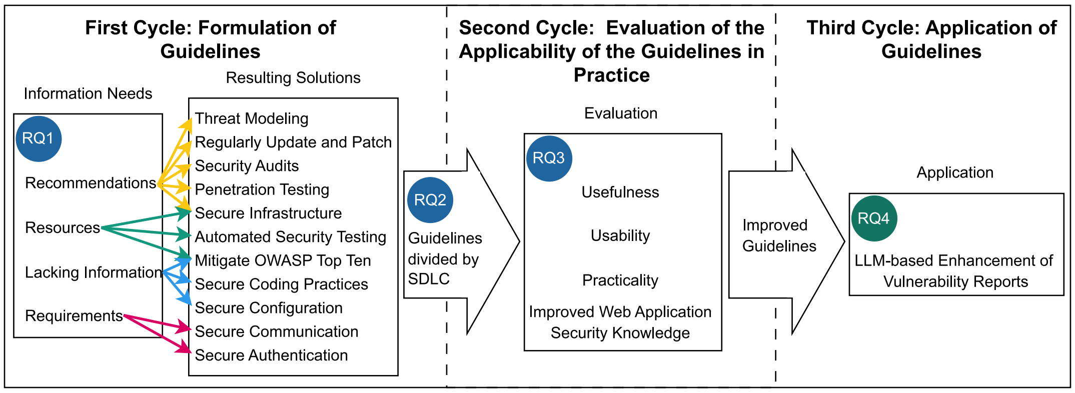

FIGURE 1 | Overview of the three cycles we performed within design science research.

TABLE 2 | Overview of the interviewees in the first cycle.

#	Exp.	Interviewee job title
1	> 10 years	Consultant Manager, QA, and Agile Coaching
2	> 10 years	Security Engineer and Penetration Technician
3	1–2 years	DevOps Engineer
4	1–2 years	Developer
5	2–5 years	Penetration Tester
6	5–10 years	Penetration Tester and Manager
7	5–10 years	Architect
8	2–5 years	Front-End Developer
9	> 10 years	Penetration Tester
10	1–2 years	Developer

specifically [64–71], as well as technical reports like OWASP or MITRE. A visual model in a tabular format was created.

To evaluate the solution, we performed an internal evaluation workshop in which four authors participated. In this workshop, the first cycle of guidelines was discussed, with a focus on making the guidelines more accessible, practical, and usable.

3.2 | Second Cycle: Evaluation of the Applicability of the Guidelines in Practice

As a result of the evaluation of the first cycle, some problem areas of the artifact were found, such as too much information and a low level of intuitive design. Thus, we performed an additional literature search to support the design and content decisions of the guidelines.

We discussed our guidelines in the context of the additionally reviewed literature (cf. Section 3.1.2) and scrutinize them based on their comprehensibility, overlap with other guidelines, and possible duplicates. Based on this some guidelines were removed or merged due to insufficient backing data or level of importance. A website representation of the guidelines was created with all guidelines. Design improvements were made, such as aligning categories according to the SDLC process.

The evaluation of our guidelines involved a survey to gather data on the practicality, usability, and relevance of the artifact. The survey was distributed to participants within the domains of software engineering, cybersecurity, and management.

3.2.1 | Survey Design

The survey was created using Google Forms. The survey questions were designed to complement the interview questions, answer RQ3, and confirm or challenge previously gathered data for RQ1. The survey was distributed through online platforms, including LinkedIn groups within the relevant domains. The surveys were further distributed by encouraging participants to invite others within their networks to answer the survey. Moreover, our industry partner helped us to recruit practitioners from the software engineering domain, cybersecurity professionals, and management. Our respondents represented different industries, organizations, and levels of experience. We decided to include six students with varying levels of experience, as long as they had worked on web application development projects before. After all, our intention was to create guidelines that should be useful to a variety of participants—not only expert developers or people involved with cybersecurity. We received 28 responses from the domains of software engineering, cybersecurity, and management. The survey consisted of three questions to establish the demographic of the respondents, followed by Likert-scale questions to collect feedback.

The interviews and surveys were conducted following ethical guidelines for research involving human participants [72]. All data collected was kept confidential and anonymized, and identifying information was removed from the data. Our questionnaire and the corresponding answers of the participants are available on Figshare.[5]

3.3 | Third Cycle: Application of Guidelines

In the third cycle, we wanted to see the applicability of these guidelines in LLM-assisted development. In general, we set up a GPT-4.1 model and design our prompts to include the Secure Web Application Guidelines (SWAG) resulting from the second cycle. Then, we prompt the LLM to enhance a given vulnerability report by our guidelines and evaluate the outcome in terms of whether the detected vulnerabilities fit the considered guideline/s and if the LLM provides insights that can help prevent the vulnerability. Consequently, we needed to construct a dataset with web applications and their corresponding vulnerabilities located as ground truth.

We decided to use well-known open-source web applications on GitHub written in PHP, as recent statistics showed that 74% of websites whose server-side programming language is known use PHP as the main programming language [13]. As a result, we selected five open-source PHP-based web application projects; then, we use Aikido (cf. Section 2.3) security scanner to identify vulnerabilities.[6] We observed that the SAST functionality provided by Aikido identified more vulnerabilities in the considered projects after each run. Thus, we ran the security scanner on the projects until the number of identified vulnerabilities converged and did not change for 10 consecutive runs of the analysis.

The Aikido scanner identified a total of 2609 vulnerabilities that were distributed across 24 vulnerability categories (cf. Table 3), as well as the code snippet that includes the vulnerability. In our experiments, these vulnerabilities and their corresponding code snippets represent the factor for our experiments. The other independent variables, namely our prompt, the used LLM, and the settings of its parameters, are not changed to test our theory that LLMs are capable of meaningfully enhancing vulnerability reports with our guidelines. The dependent variable in our experiments is the output of the LLM, the quality of which we have assessed by security experts.

To prepare for the LLM evaluation, we construct a smaller dataset of 47 data points, each consisting of the name of the affected file, the identified vulnerability, and the vulnerable code snippet. We ensure the diversity and the representativeness of the dataset by randomly selecting two entries from the Aikido vulnerability report for each of the 24 vulnerability types (cf. Table 3), resulting in 47 data points (one vulnerability type appeared only once). We also decided to extend the code snippets containing the vulnerabilities to add more context, by adding two lines of code before and after the vulnerable lines of code.

Finally, we ran our model on the dataset with 47 data points three times by sending a message object to the OpenAI API. The object included the name of the model, that is, GPT-4.1, the temperature, the system prompt, and the user prompts. Temperature is a parameter that controls the randomness of the model output. We set it to 0.2 because this low value is commonly used for code generation, as it produces more consistent and deterministic responses [73, 74]. The system prompt sets the overarching role of the model, while the user prompt contains specific instructions or questions. Therefore, we placed our guidelines in the system prompt to ensure the model always follows them regardless of the individual user inputs. The user prompt implemented

TABLE 3 | Overview of the reported vulnerability types from the Aikido vulnerability report for all projects.

Vulnerability type	Number of occurrences
Prototype pollution vulnerability detected	1
Potential SQL injection via Laravel function	2
SSL certificate verification turned off during requests	2
Using potentially unsafe FTP connections to move data	2
HTTP request might enable SSRF attack	3
Path traversal attack possible via file functions	3
Rendering unescaped input in HTML template can lead to XSS attacks	3
Template Injection in GitHub Workflows Action	3
Rendering unescaped input can lead to XSS attacks	4
Using phpinfo() can expose sensitive info to users	4
Using unsafe GitHub Actions trigger may allow privilege escalation via CI/CD	6
A timing attack might allow hackers to brute-force passwords	8
Using backticks in PHP can lead to remote code execution	8
NoSQL injection attack possible	9
Using var_dump() can expose sensitive info to users	11
Remote Code Execution possible via eval()-type functions	16
Using v-html in Vue templates can lead to XSS attacks	20
Unsafe eval usage can lead to remote code execution	23
Third-party GitHub Actions should be pinned	49
Unsafe exec usage can lead to remote code execution	56
Using unserialize can lead to remote code execution	114
Using document write methods can lead to XSS attacks	117
Potential SQL injection via string-based query concatenation	757
Rendering unescaped input in handlebar/mustache template can lead to XSS attacks	1388

```
Given the following code snippet from rss.php, which contains the vulnerability "using unserialize"
    can lead to remote code execution. Which SWAG guidelines should be followed to prevent this
    vulnerability in the future? Quote only the corresponding SWAG guidelines that should be
    followed to prevent this vulnerability in your answer and explain your rationale behind the SWAG
     guidelines for the given code snippet.

        function unserialize ( $data ) {
        return unserialize( $data );
     }

Your answer should be structured as a JSON object with the fields swag_guidelines_applied and
    swag_guidelines_rationale and structure swag_guidelines_rationale using the SWAG guidelines that
     should be followed to prevent this vulnerability in the future.
```

LISTING 2 | Example prompt to map a Unserialize Vulnerability to SWAG guidelines.

a prompt template that specifies the vulnerable code and the relevant vulnerability type, instructions to provide the relevant guidelines to prevent the vulnerability as well as how the LLM output should be structured in a JSON format. An anonymized example of one of our user prompts is shown in Listing 2. We also employed the chain-of-thought prompt technique to obtain a detailed reasoning of the identified vulnerabilities.

Because we ran the model three times, we observed some variations in the output for the same input due to the nondeterministic nature of LLMs. To account for this, we treated these results as additional data points resulting in a total of 73 data points.

The different output variations arise not only from the nondeterminism of LLMs but also from the fact that vulnerabilities can be prevented by various measures. For example, for the same code snippet affected by the vulnerability "Using document write methods can lead to XSS attacks," the guidelines recommended by the LLM include sanitizing input, using secure coding standards, and additional information on Injection attacks. In a subsequent run, the output of the LLM contains the same guidelines and additional information, as well as a guideline related to performing code reviews and testing.

3.4 | Data Analysis

In this section, we describe the data analysis we performed in our study. The first and second cycles are concerned with specifying and evaluating our guidelines. Thus, we present our data analysis for both cycles combined in Section 3.4.1, which is also part of our previous work [12]. In Section 3.4.2, we describe our data analysis procedure to investigate the possibility of enhancing vulnerability reports with our guidelines using an LLM.

3.4.1 | First and Second Cycle

The interviews were transcribed and analyzed using thematic analysis to identify key themes in the data. The surveys were analyzed using descriptive statistics to provide a quantitative perspective on the data.

The interview data was analyzed using inductive coding to identify emerging themes and patterns [75]. Two researchers independently coded the data and compared the results. Once the themes were identified, they were aggregated and discussed to arrive at conclusions about the research questions. As an example for the data analysis, we consider the following quote: "I would say that kind of depending on what kind of a web application it is, it is in the developers' interest to keep it secure, but in some cases, the benefit does not outweigh the cost." The quote talks about the cost–benefit tradeoff connected to security. It was coded with "cost" and "decisions" in the "management" theme.

We used descriptive statistics to analyze the survey responses. Concretely, the data were analyzed using measures such as frequency distributions, central tendencies, and variability to describe the characteristics of the sample and the distribution of responses to the survey questions.

The interview and survey data were integrated to provide an understanding of the guidelines necessary for software engineers to develop secure web applications. The qualitative and quantitative data were triangulated [76] with related literature, interviews, and survey data. The interviews mainly provided information regarding domain knowledge and present issues, whereas the surveys primarily confirmed assumptions based on the interviews and evaluated the artifact.

3.4.2 | Third Cycle

The results generated by the model were independently evaluated by three experts. Two of these experts are external researchers who are not involved in this research. The first expert (E1) is a researcher in the range of 2–5 years of cyber security experience. The second expert (E2) is a researcher in the range of 5–10 years of cyber security experience who also has previous industrial experience. The third expert (E3) is one of the authors of this work, who has about 2 years of experience in security modeling. All evaluators are seasoned computer scientists with programming experience.

Three aspects were evaluated for each result data point: (i) Are the guidelines proposed by the LLM relevant to the code snippet and vulnerability? (Relevance) (ii) Do the explanations generated by the LLM match the content of the corresponding guidelines? (Explanations match guidelines) (iii) Are the proposed guidelines helpful in preventing the corresponding vulnerability in the future? (Help in preventing vulnerability).

For each generated LLM output, the evaluators received the anonymous absolute path to the affected file in the project, the anonymized affected code snippet, our guidelines proposed by the LLM for the respective vulnerability, and the explanations generated by the LLM. The evaluators should document which guidelines and explanations do not fulfill the above-mentioned aspects. Results that correctly fulfilled all aspects were marked with y in our data set by the evaluators. For each entry, the evaluators had the opportunity to document their own comments. In addition, they analyzed the different outputs generated from the same prompt across the three runs to assess whether the variations were meaningfully different.

The results of the expert assessment were aggregated and we report them in Section 4.4. We provide the anonymized expert assessment of our results as Supporting Information. Please note that we cannot share the project names or the respective vulnerable code snippets due to ethical considerations.[7]

3.5 | Threats to Validity

We discuss threats to validity using Easterbrook et al.'s criteria [77].

3.5.1 | Internal Validity

The data we collected in interviews, surveys, and the practitioners' assessment in our experiments are based on self-reported information, which may introduce bias. To reduce the likelihood of bias, the interview questions were designed to be clear and nonleading. Two interviewers were present to collect data, which reduces the impact of interviewer bias. The interview questions were reviewed by the four participating researchers. Anonymity and nonevaluation of answers were ensured to encourage truthful responses. We collected data using interviews as a qualitative data source and a survey as a quantitative data source. This approach allowed us to triangulate the findings, further improving internal validity. Additionally, although the interviewees might not have been aware of best practices in cybersecurity, they possessed sufficient knowledge to understand the questions and provide informed responses. This ensured the data collected were still relevant and valuable for the study. The results of our experiments were evaluated by three experts independently and subsequently compared to ensure correctness and reduce the likelihood of bias.

We used the SAST functionality of Aikido to generate the vulnerability reports used in our experiments. Static analyses are often prone to generating a large number of false positives. To address this weakness, we have checked whether the flagged vulnerabilities are contained in the corresponding code snippets that we used in our experiments.

Finally, the nondeterminism of the outcomes is inherited from the randomness in LLM-based experiments. To address this threat, we control the LLM temperature and set it to 0.2, reducing the randomness of the generated content. We also ran the LLM over our dataset three times and manually inspected the results. In general, we did not observe any major variations in the explanations across runs.

3.5.2 | External Validity

The findings resulting from the first and second cycle are not fully generalizable to other software development contexts, as the study was conducted with a specific population and geographic location. Efforts have been made to include a variety of roles, but the sample might not fully represent the populations being studied [78]. We decided to include less experienced participants, as long as they were familiar with web application development. We wanted to design guidelines for a broad audience—not only expert developers or developers who were familiar with cybersecurity issues. Some of the involved survey respondents were software engineering students, who can be seen as junior software engineers. It would be beneficial to run further evaluations with more experienced practitioners.

In our experiments (cf. third cycle), we used data obtained from well-known web applications that were mainly implemented in PHP because most websites still use PHP. However, we do not assume that this limits the generalizability of our results, as we have taken care to consider at least two representatives per vulnerability in our experiments. Additionally, the vulnerability reports, which we used in our experiments, also contain vulnerabilities, for example, in JavaScript and TypeScript code.

3.5.3 | Construct Validity

The interview and survey questions may not have fully captured the constructs being studied. To address this, the terminology used in the questions was explained to the participants, considering that terms such as “OWASP” may not be widely known in the specific context.

The survey questions may not accurately measure the construct of interest (usability and helpfulness of the artifact). There is also a risk that important aspects have not been covered by the survey questions.

We do not evaluate the completeness of the proposed guidelines in our experiments as it can be argued that it is best to always follow all guidelines. For example, using secure coding standards always makes sense to prevent vulnerabilities. In addition, different guidelines can prevent the same vulnerability. For example, “Secure Communication” and “Secure Configurations” recommend using SSL/TLS. Recommending both would just generate unnecessary noise for the developers.

3.5.4 | Reliability

We provide Supporting Information to enable others to replicate this study. The results of the survey may have been influenced by factors such as the timing of the survey, the way we asked the questions, or the participants' level of engagement. Additionally, the tool being evaluated in the survey is one made by the authors for this research. The questions we asked were framed in a mainly positive manner, potentially producing biased answers. These factors should be considered when interpreting the results.

Two external practitioners, who are not involved in this study, and one author evaluated independently the LLM output to strengthen the reliability of our results.

4 | Findings

In the following section, we present the result of our interview study and highlight the identified information needs (RQ1). In Section 4.2, we present our guidelines (RQ2), followed by the practitioners' assessment of their applicability (RQ3). Both sections present work that is already published in our previous work [12]. Finally, in Section 4.4, we present the expert assessments showing to what extent our guidelines are suitable to enhance vulnerability reports meaningfully to prevent the same vulnerabilities in the future (RQ4).

4.1 | RQ1—Security Information Needs

In this Section, we present our findings related to RQ1: What are the security information needs of software engineers to develop secure web applications?

The findings from the conducted interviews revealed problems related to cybersecurity education. Firstly, there is a significant lack of education on cybersecurity, with many developers reporting that it is not covered satisfactorily in their education, job descriptions, or workflows. One of the penetration testers stated that

> [...] you're not taught how to build it securely or how to even in any way at all mitigate these kinds of vulnerabilities that are actually quite common.

Another developer mentioned that

> I don't think I have the knowledge right now to identify these potential issues or vulnerabilities that might happen from the code I write.

Nevertheless, they mention that security does come to mind when developing. This lack of security is further exemplified by one of the Cybersecurity Managers, saying

> I know that despite that there's awareness that there's vulnerabilities, the work done to mitigate these is not sufficient.

The lack of appropriate resources that explain cybersecurity concepts in a way that developers can understand or apply is also a common challenge. Many developers are unaware of resources such as OWASP that are well-known among cybersecurity professionals. One security engineer interviewee mentions:

> I would say the OWASP model but then again it requires them to actually have the time and the interest to understand the model.

Regarding the question, what they think is the most important to think about when developing a secure web application, further confirming the importance OWASP but exemplifying the potential issue of information being too complex. To confirm this, one of the developers was asked about OWASP, they stated:

> I'd say it's quite useful. But I just haven't found any sort of practical application for it just yet. Because I'm not too experienced with the resource itself.

We found that the reuse of insecure code found online is a prevalent issue among developers. A penetration tester mentioned:

> if you find an example code somewhere like let's say Stack Overflow without even validating that the code is secure enough, you're basically implementing a code that might be vulnerable without even knowing about it.

This practice increases the risk of common vulnerabilities, as developers may not know how to properly implement certain features or what to look for in terms of security vulnerabilities, including those outlined in the OWASP Top 10.

Inadequate access to cybersecurity professionals and a lack of security culture within development teams are also identified challenges, as highlighted by the two statements:

> [...] there's a huge gap in both ways because the company I'm working for has developers and we have a security team. And I realize that there's actually a big gap between us, even if we're working for the same company. I'd say the overall level of security awareness when it comes to companies and developers is quite low.

Developers may not know where to look for information or have limited access to cybersecurity experts who can provide guidance.

Securing user data and backend systems are also recognized as critical concerns. An interviewee stated:

> [...] guidelines should be at the bare minimum. At least make sure that the code you write doesn't put your [...] users' data at risk.

Developers acknowledge the importance of these areas but do not always know how to effectively address them.

Summary

Based on the findings from the interviews and literature review, the following are the information needs of software engineers to develop secure web applications:

1. Education and training on cybersecurity, to help understand mitigation strategies and key concepts.
2. Information on resources that explain a way that developers can understand and apply.
3. Guidance on securing user data and backend systems.
4. Awareness of vulnerabilities and threats, such as the OWASP Top 10.
5. Importance of secure coding practices, including avoiding the reuse of insecure code found online.
6. Security culture within development teams and adequate access to cybersecurity professionals who can provide guidance and support.

4.2 | RQ2—Guidelines

In this section, we present our findings related to RQ2: What potential guidelines can support software engineers in developing secure web applications?

The findings suggested that potential guidelines can provide software engineers with the necessary information to develop secure web applications. Utilizing such guidelines and promoting a security culture within development teams can help raise awareness and emphasize the importance of cybersecurity among software engineers. We found that presenting cybersecurity concepts, practices, and resources in a more accessible way for developers indicates the possibility of improving their understanding and chances of using secure coding practices (cf. Figures 2–4).

The guidelines created from problem identification and interviews were condensed and coupled with backing literature and sources. Our data indicated that accessibility and ease of understanding were the main requirements for our artifact. Therefore,

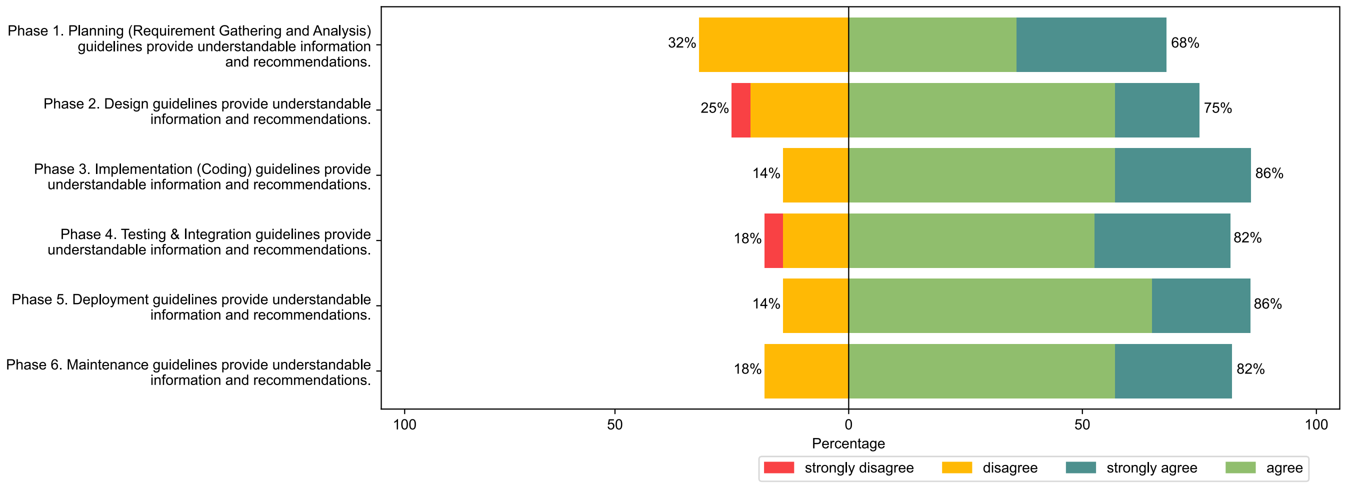

FIGURE 2 | Responses to our Likert-scale questions ($n = 28$) regarding the participants' assessment of the understandability of our guidelines per Software Development Life Cycle phase.

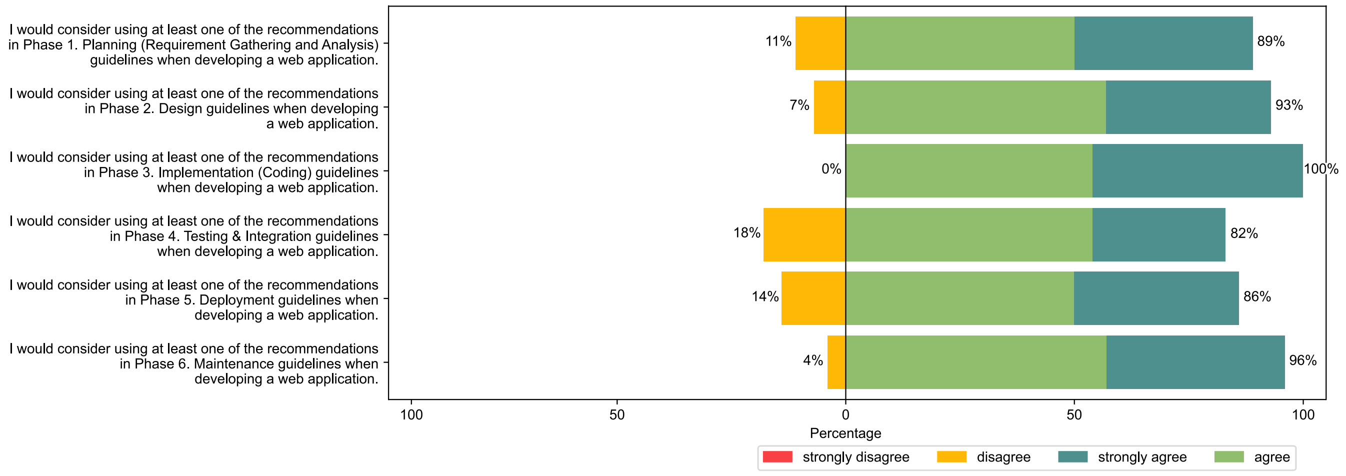

FIGURE 3 | Responses to our Likert-scale questions (n=28) regarding the participants' likelihood to consider our guidelines per Software Development Life Cycle phase.

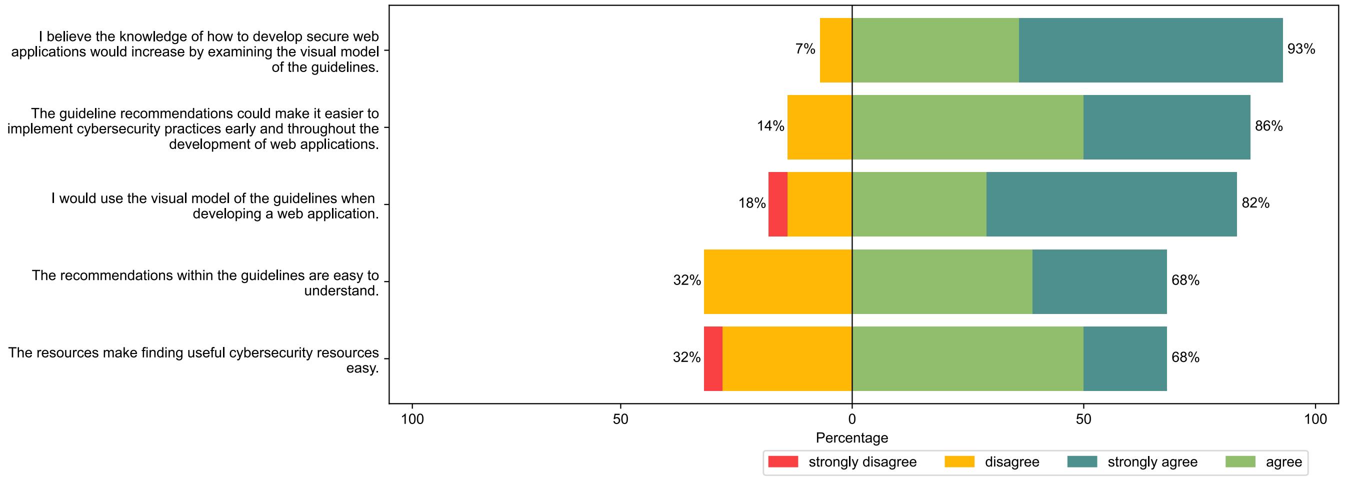

FIGURE 4 | Responses to our Likert-scale questions ($n = 28$) regarding the participants' overall assessment of our guidelines.

TABLE 4 | Tabular representation of the guidelines categorized according to the phases of the Software Development Life Cycle.

Phase	Technique
Planning	Threat modeling
Design	Secure communication and secure authentication
Implementation	OWASP Top 10, secure coding practices, and penetration testing
Testing and integration	Automated security testing
Deployment	Secure configuration
Maintenance	Regular update and patch and security audits

a website representation of the guidelines was created.[8] It can simplify information and improve retention, as many of the existing sources include technical concepts that may be difficult to understand solely through written text. By using visual elements, the guidelines can be simplified and presented in a more visually engaging way, making it easier for readers to grasp the key concepts and remember them, along with finding them easily. Table 4 shows an overview of the guidelines categorized according to the SDLC process. We will explain these in more detail in the following.

4.2.1 | Threat Modeling

Many interviewees in the cybersecurity domain recommended threat modeling as a useful tool. When cybersecurity management was asked what advice they would give to developers, they answered:

> it's at tops [a] two to three days session where we sit down and go through what their solution is going to look like.

Another penetration tester stated:

> I love threat modeling and that's something I also recommend going back to what I said before about being part of the projects from the beginning.

Similarly, Myagmar et al. [71] also recommend threat modeling by discussing how it can be used as a foundation for specifying security requirements.

4.2.2 | Secure Communication

If exchanges are not secured through secure communication channels, web applications might have sensitive data leaked or compromised. Our findings suggest that a large issue that comes with web applications is the exchange of sensitive information or data. According to the developers interviewed, communication channels are one of the most known potential security risks. A penetration technician also mentioned that using encrypted communications protocols is one of the most important features for potential tools aiding developers in securing their web applications. However, developers may not always know how to implement or maintain it; therefore, it should be one of the recommendations in the guidelines.

4.2.3 | Secure Authentication

When discussing the security levels of projects that the interviewees have worked on, an architect responded with the security being low to medium-low and authentication being "probably the biggest problem." Another interviewee stated that a big part is to

> be aware of the users that will be using your application, what their sort of normal level of technical experiences, isolate your components and secure them through authentication or authorization methods.

Authentication is clearly a concern that developers should be more aware of when developing, and therefore, it is present in the guidelines.

4.2.4 | OWASP Top 10

OWASP was brought up in several interviews, primarily by those working within cybersecurity as penetration testers or management. While OWASP is well known within the domain of cybersecurity, it is rarely known within software engineering. Fifty percent of our respondents strongly disagree with being familiar with the OWASP Top 10. Therefore, we see that there is a need for increased awareness regarding resources such as OWASP. As one penetration tester mentioned:

> the recommendation is to follow OWASP, use OWASP as your main go-to reference when it comes to designing your application.

4.2.5 | Secure Coding Practices

Establishing and using secure coding techniques is essential for creating secure web applications. It is also crucial when a developer assesses whether someone else's code is secure. This encapsulates most of the issues present in the security of web applications, and developers may lower the risk of vulnerabilities and attacks by adhering to secure coding standards and recommendations, such as those provided by OWASP. The practice of secure coding guidelines has previously been found to be impactful when it comes to the quality and security of products [79]. Similarly, when a manager was asked what it means for cybersecurity in a project to be up to date, one of the things they look at is:

> Do they actively review code from a security standpoint and both correct code that's insecure and also rewrite guidelines for secure coding?

4.2.6 | Penetration Testing

In order to create secure web applications, penetration testing can be an important measure as it reveals weaknesses that could go unnoticed throughout the development process. Penetration testing is often performed as part of the implementation phase and thought of before other testing activities start. One penetration tester mentioned:

> even before starting to developing the application [...] you can involve security consultants or penetration testers.

Though this may not be possible in all projects. The recommendations of penetration testing could vary depending on company size and resources, though automated web penetration tests and tools should easily be implemented to help reveal risks and vulnerabilities.

4.2.7 | Automated Security Testing

Automated security testing can assist in quickly and effectively identifying security vulnerabilities and is a crucial component of creating secure web applications. Integrating automated testing also allows for saving both time and efficiency when developing and deploying web applications. As one manager suggested during their interview to perform "at least some basic security testing." Automated security testing is already an established way of ensuring continuous security [66] and is thus present in the guidelines.

4.2.8 | Security Configuration

By configuring web servers, databases, and other components according to secure configurations, developers can reduce the attack surface of web applications and make them more resilient to threats. One of the developers mentioned that the company they work for is making sure the security of the projects is up to date by "All of our internal-external projects share the same security configurations" and continues by mentioning the implementation of regular security checks. This is something the respondent considered to work well and showcased that the company was communicating those routines to developers. In the OWASP Top 10 in 2021, security misconfiguration is at the fifth spot of most significant security threats for web applications, and should thus be taken into more consideration during development.

4.2.9 | Regularly Update and Patch

By regularly applying security updates and patches on the software and components of their web applications, developers can mitigate some of the most common threats and attacks. One developer described a scenario in which they had to update dependencies and packages. They stated:

> for example, the react app is not really good at updating the dependencies. So the whole package is really outdated, to be able to outcome that we need to add some resolutions individually.

To mitigate these issues, they remove unused dependencies and continuously inventory the versions and dependencies within the applications. Another developer interviewed mentioned that the current practices of regularly updating include the following:

> we use Ansible to configure everything so everything runs on the same version of a library runs on the same OS image. They all get the same updates and that helps us keep things consistent.

Because it is such a common problem, it is included in the guidelines.

4.2.10 | Security Audits

Security audits seem to be common within the industry, one of the developers interviewed mentioned that they are using Yarn audit[9] to detect if there are any major security threats. One penetration tester also mentioned security audits when asked what level of cybersecurity they would rank the projects they have worked on, they then mentioned that customers they work with often have to adhere to using security audits:

> Due to the fact that they actually have some regulations controlling them, they won't be allowed to do stuff without security audits.

By regularly conducting security audits such as vulnerability assessments and code reviews, developers can find and fix potential weaknesses. Security audits are a common countermeasure to cybersecurity attacks and should therefore also be part of the guidelines.

Summary

We created guidelines for developing secure web applications, as listed in Table 4.

4.3 | RQ3—Evaluation of the Applicability of the Guidelines in Practice

In this section, we present our findings related to RQ3: How can those guidelines be applied to the development of secure web applications?

Our interviews suggest providing clear mitigation strategies for common vulnerabilities and threats are key points in enabling software engineers to proactively address security concerns during the development process. Most interviewees stated that integrating cybersecurity as early as possible can potentially make the process less of an issue later on. We found that to make cybersecurity more understandable for software engineers, it is useful to consider the SDLC phases to make it clear to software engineers when and where they need to be aware of certain cybersecurity measures.

The current practice of integrating cybersecurity primarily involves hiring cybersecurity professionals or other means when problems arise after deployment [80, 81]; therefore, there should be a focus on integrating it earlier and throughout. Additionally, it is largely agreed among the cybersecurity professionals interviewed that cybersecurity practices should be present in all stages of the software development process, when asked what role cybersecurity plays a cybersecurity manager mentioned:

> It does play a role and needs to be done early and also continuously throughout the development process.

To make cybersecurity more understandable for software engineers, a common methodology such as SDLC allows for the breakdown into different phases to make it clear to software engineers when and where it may be applicable to perform or be aware of certain cybersecurity measures.

All developers interviewed mentioned utilizing some sort of development process, largely some sort of agile adaptation:

> The development process is usually guided by the company that I work for. And at present, it's an Agile Scrum approach that we are using.

Given that the SDLC phases in the guidelines can be mapped to pretty much any development process, they are considered to be generally applicable by our participants. By applying guidelines into phases of the SDLC, users can obtain specific security requirements and recommendations that are relevant to the stage they are currently at in development and easily applicable to most development processes. Additionally, this will potentially make it more straightforward for software engineers to learn to apply secure coding practices, conduct thorough security testing, and ensure the application is deployed and maintained securely. This assumption is backed up by our results, which show 89.3% of participants agreed or strongly agreed that breaking up the recommendations into SDLC provides an understanding of when certain activities could take place when developing secure web applications.

Figure 2 shows the survey responses regarding the participants' assessment of the understandability of our guidelines per SDLC phase. It can be seen that the participants generally assess the understandability of our guidelines per SDLC phase very positively. Overall, the developers are positive about our guidelines, as can be seen in Figure 3, which shows that over 80% of the survey participants would consider at least one guideline per SDLC phase.

Figure 4 shows our survey responses regarding the participants' overall assessment of our guidelines. Figure 5 provides an overview of our survey participants, showing the number of participants by their experience and their work domain. The demographics of the survey participants can also be found in the Supporting Information. Our participants had varying levels of experience; 57.1% of the participants worked in Software Engineering (Developer, UX, Architecture, etc.). The remaining participants worked in Cybersecurity, Management, Architecture, or System Administration. The survey results were overall positive with most respondents being unanimous.

When discussing the survey results, we refer to mean and standard deviation as measures of central tendency and dispersion, respectively. The values ranged from 0 to 5. A mean of 2.5 or less is considered a negative response, and a mean of 2.5 or greater is considered a positive response. The statement "I am familiar with cybersecurity" resulted in a mean of 2.54 with a standard deviation of 1.04, indicating a moderate

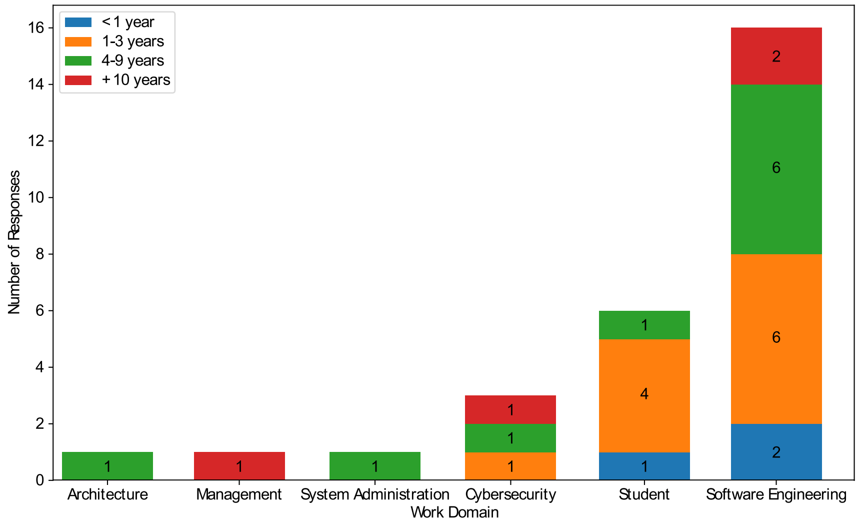

FIGURE 5 | Overview of the survey participants.

level of familiarity with cybersecurity. Answers to the statement "I find it difficult to identify security issues when developing web applications" resulted in a mean of 2.32 and a standard deviation of 0.82 showing that respondents did find it difficult to identify security issues. However, they had a moderate level of understanding of where to find information about securing web applications (with a mean of 2.61 and a standard deviation of 0.92). Respondents found it difficult to understand the information given in various sources regarding developing secure web applications (with a mean of 2.21 and a standard deviation of 1.03). This indicates that there needs to be a more understandable source of information for software developers.

Answers to the statement "I wish there was a site that collected the most common cybersecurity practices and sources for secure web application development." resulted in a mean of 3.57 and a standard deviation of 0.84. This indicates that presenting the guidelines in a website format was reasonable. They also believed that web development would benefit from guidelines to improve security.

Overall, the survey results indicate that participants found the guidelines to be useful in developing secure web applications, especially when the guidelines are presented as a website and when broken down by the SDLC. The survey indicated that participants found written guidelines less compelling than an interactive and visually compelling website.

Summary

We found that our guidelines are perceived as applicable, especially if they are integrated into the stages of the SDLC. By breaking down the recommendations into the SDLC phases, developers can obtain specific recommendations that are relevant to the stage they are currently at.

4.4 | RQ4—Enhancing Vulnerability Reports

In this section, we present our findings related to RQ4: To what extent can those guidelines be leveraged to improve the security of web applications with LLMs?

Table 5 shows an aggregated overview of the proposed guidelines (cf. Section 4.2) per vulnerability type. A detailed overview of the guidelines proposed by the LLM and the generated explanations can be found in our Supporting Information.[10]

Overall, our results from the expert evaluation are very positive. As already described in Section 3.4.2, the three experts evaluated the data points considering three aspects: (i) relevance, (ii) explanations match guidelines, and (iii) help in preventing vulnerability.

The first external expert (E1) rated 71 of the 73 data points to be evaluated as completely correct, in the sense that they fulfill all three aspects. The second external expert (E2) rated 66 of the 73 data points to be evaluated as completely correct, in the sense that they fulfill all three aspects. The evaluation of the datasets by the internal expert (E3) comes to similar results. Of the 73 data records to be evaluated, the internal expert rated 69 as completely correct in terms of the three aspects evaluated. In addition, the internal expert rated four contributions as partially correct.

The only data points the external expert E1 rated negatively were related to the vulnerability "Using phpinfo() can expose sensitive info to users." For this data point, the external expert E1 concluded that the suggestions would help to prevent the vulnerability in the future (evaluation aspect iii) but that performing a full risk analysis to identify a known insecure function is too excessive.

The second external expert E2 assessed seven data points as partially correct. For five data points, the expert E2 assessed that a guideline suggesting the implementation of error handling,

TABLE 5 | Aggregated overview of the guidelines recommended by the LLM for each type of vulnerability.

Vulnerability type	Threat modeling	Secure communication	Secure authentication	OWASP Top 10	Secure coding practices	Penetration testing	Automated security testing	Security configuration	Regularly update and patch	Security audits
Prototype pollution vulnerability detected		✓		✓	✓					
Potential SQL injection via Laravel function		✓			✓					
SSL certificate verification turned off during requests		✓			✓			✓		
Using potentially unsafe FTP connections to move data		✓						✓		
HTTP request might enable SSRF attack				✓	✓	✓		✓		
Path traversal attack possible via file functions				✓	✓			✓		
Rendering unescaped input in HTML template can lead to XSS attacks		✓		✓	✓					
Template Injection in GitHub Workflows Action		✓			✓			✓		
Rendering unescaped input can lead to XSS attacks		✓		✓	✓					
Using phpinfo() can expose sensitive info to users	✓			✓	✓	✓		✓		✓

(Continues)

TABLE 5 | (Continued)

Vulnerability type	Threat modeling	Secure communication	Secure authentication	OWASP Top 10	Secure coding practices	Penetration testing	Automated security testing	Security configuration	Regularly update and patch	Security audits
Using unsafe GitHub Actions trigger may allow privilege escalation via CI/CD	✓			✓	✓	✓		✓		✓
A timing attack might allow hackers to brute-force passwords		✓	✓	✓	✓					
Using backticks in PHP can lead to remote code execution				✓	✓					
NoSQL injection attack possible					✓					
Using var_dump() can expose sensitive info to users		✓		✓	✓					
Remote Code Execution possible via eval()-type functions				✓	✓					
Using v-html in Vue templates can lead to XSS attacks				✓	✓					
Unsafe eval usage can lead to remote code execution				✓	✓					
Third-party GitHub Actions should be pinned		✓			✓			✓	✓	
Unsafe exec usage can lead to remote code execution				✓	✓					

(Continues)

TABLE 5 | (Continued)

Vulnerability type	Threat modeling	Secure communication	Secure authentication	OWASP Top 10	Secure coding practices	Penetration testing	Automated security testing	Security configuration	Regularly update and patch	Security audits
Using unserialize can lead to remote code execution				✓	✓					
Using document write methods can lead to XSS attacks				✓	✓					
Potential SQL injection via string-based query concatenation					✓					
Rendering unescaped input in handlebar/mustache template can lead to XSS attacks				✓	✓					

which is part of secure coding practices, does not help to prevent the four vulnerabilities "Unsafe exec usage can lead to remote code execution," "Unsafe eval usage can lead to remote code execution," "Rendering unescaped input can lead to XSS attacks," and "Template Injection in GitHub Workflows Action" in the future. Additionally, in relation to the vulnerability "Prototype pollution vulnerability detected," the expert E2 assessed that the suggestion of the LLM that prototype pollution results from insecure design which is part of OWASP Top 10 does not help to prevent it in the future. Regarding the vulnerability "Using backticks in PHP can lead to remote code execution," the expert E2 assesses that the suggested use of secure APIs which is part of secure coding practices is not relevant to the code and the vulnerability it contains.

The internal expert E3 assessed four data points as partially correct, as the guidelines proposed by the LLM for these data points are only relevant under specific conditions that were not apparent from the code examples. For example, for the vulnerability "Prototype pollution vulnerability detected," the LLM suggested twice to follow the guideline to sanitize input, although the corresponding code snippet is too generic to determine whether it is handling input. The other two data points concern the vulnerability "Path traversal attack possible via file functions." Here, the internal expert E3 assessed with the same reasoning that the points "Broken Access Control," "Security Misconfiguration," and "Injection" listed under Mitigate OWASP Top 10 security risks only make sense in specific scenarios and these are not evident from the code.

We received further insights from the experts through the optional comment field. The two experts E1 and E3 criticized the high level of abstraction of the guidelines which makes it hard for developers to apply them in practice. To address this issue, the external expert E1 suggested generating code snippets with suggested fixes implemented alongside the explanations and for the LLM to reference the code snippet passed in the prompt in its generated explanations.

Summary

We found that our guidelines can be leveraged to improve the security of web applications by enhancing vulnerability reports with the help of an LLM. However, the high abstraction level of our guidelines might make it difficult for software developers to translate the guidelines directly into practice.

5 | Discussion

Our findings provide important empirical evidence on the gap between developers and cybersecurity experts. They also indicate what techniques are prioritized today and where better support is needed. In particular, our results shed light on developers' information needs (RQ1). The findings partially confirm developers' lack of security knowledge [69] and align with the findings of Gasiba et al. [68] that despite the lack of secure coding policies awareness among the developers, the majority acknowledges its importance. Kalhoro et al. [7] found that several social, personal, and technological factors influence cyber hygiene practices. While their findings confirm some of ours, they do not stress the issue of code reuse and its potential impact on the security of a system. This might be because, in web application contexts, it is more common than in other domains to reuse code from online sources. Moreover, because the introduction of LLMs, developers have leveraged these tools for various tasks, including web development. Tóth et al. [21] found that websites created by GPTs are vulnerable which adds to the problem of reusing online code. We believe that the need for secure coding guidelines is increasing along with the increased use of AI for coding. Our participants also acknowledge the importance of secure coding guidelines and of cybersecurity in general and consider it to be a high-priority concern. Hence, our results do not confirm the findings of Kalhoro et al. [7] that cybersecurity was not prioritized because of budget constraints.

There are some studies [82–84] that already focus on security guidelines. However, these studies usually focus only on the development phase (RQ2). In addition, these studies evaluate the guidelines only in terms of their effectiveness or their degree of application in practice, but neglect the opinions of developers (RQ3). This could also be the reason why security guidelines are not very popular in practice, as their practicality has not been sufficiently investigated.

Our findings indicate a need for further research to identify and assess alternative approaches for improving web application security. We found that developers struggle with finding starting points to improve their cybersecurity practices and developing lightweight techniques for threat modeling is a promising area of future work to address that challenge. Our results in Section 4.4 indicate that LLMs are capable of meaningfully enhancing vulnerability reports with our guidelines (RQ4). Providing a basis for a promising integration of our secure coding guidelines into IDEs or other tools, so that regular updates, audits, testing, and analysis activities can be performed. However, the feedback from the experts evaluating our experiment results shows that some of our guidelines are too abstract, making it hard for developers to apply our guidelines directly in practice. Therefore, we plan to investigate whether this limitation can be overcome by using an LLM to generate code examples that showcase a possible fix for a vulnerability or whether we need to refine our guidelines.

6 | Summary

We performed 10 semistructured interviews with practitioners from the same company and a survey with 28 responses, of which six stem from students, to design a set of guidelines that aid software engineers in developing secure web applications. The results of our study show that over 80% of the survey participants would consider at least one of our guidelines in each development phase. Additionally, the majority of our survey participants agreed that our guidelines provide understandable information and recommendations for each phase of the SDLC.

Additionally, we performed experiments that examined whether an LLM can enhance vulnerability reports with recommendations comprising our guidelines to support developers in preventing these vulnerabilities in the future. Overall, the results of our experiments show the feasibility of meaningfully enhancing vulnerability reports with our guidelines. Based on this insight, we plan to extend our experiment setup and examine to what extent we can tailor the LLMs recommendations, even more, to fit identified vulnerabilities by using custom LLMs or by applying retrieval-augmented generation (RAG), a technique that enhances model outputs by incorporating relevant external information. We also plan to extend our initial data set by considering more data records from web applications that were implemented using other programming languages than PHP.

In addition, revising and adapting the guidelines to include practices for using LLMs in web development is an important future work. It would also be interesting to conduct a longitudinal study of the solution presented, where developers would use the guidelines for an extended period. This would allow concrete observations of the effects of working with the guidelines on their security knowledge levels.

Author Contributions

Raffaela Groner: conceptualization, methodology, software; validation, investigation, writing – original draft, visualization, project administration. **Klara Svensson:** conceptualization, methodology, software, validation, formal analysis, investigation, writing – original draft, visualization. **Drake Axelrod:** conceptualization, methodology, software, validation, formal analysis, investigation, writing – original draft, visualization. **Ranim Khojah:** conceptualization, methodology, software, writing – original draft. **Mazen Mohamad:** conceptualization, methodology, writing – review and editing, supervision. **Rebekka Wohlrab:** conceptualization, methodology, writing – review and editing, supervision, funding acquisition.

Acknowledgments

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

Data Availability Statement

The data that support the findings of this study are openly available in “Supplementary material” at https://figshare.com/s/4b1f0b25130a7d44a0e7 and in “Supplementary material LLM Experiments” at https://figshare.com/s/972e72967b7b9e53f63a?file=55050311.

Endnotes

[1] https://www.qestit.se/, access: July 30, 2025.

[2] https://www.owasp.org/, access: July 30, 2025.

[3] https://figshare.com/s/4b1f0b25130a7d44a0e7, access: July 30, 2025.

[4] See footnote 1.

[5] see footnote 3.

[6] All projects were downloaded and scanned on April 30, 2025.

[7] https://figshare.com/s/972e72967b7b9e53f63a?file=55050311, access: July 30, 2025.

[8] https://dev-swag.vercel.app/, access: July 30, 2025.

[9] https://classic.yarnpkg.com/en/docs/cli/audit/, access: July 30, 2025.

[10] https://figshare.com/s/972e72967b7b9e53f63a?file=55050311, access: July 30, 2025.

References

1. T. Gasiba, U. Lechner, and M. Pinto-Albuquerque, "CyberSecurity Challenges for Software Developer Awareness Training in Industrial Environments. Innovation Through," *Information Systems* 47 (2021): 370–387, https://doi.org/10.1007/978-3-030-86797-3_25.

2. H. Assal and S. Chiasson, "'Think Secure From the Beginning': A Survey With Software Developers," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Association for Computing Machinery, 2019), 1–13, https://doi.org/10.1145/3290605.3300519.

3. S. Kumar, R. Mahajan, N. Kumar, and S. K. Khatri, "A Study on Web Application Security and Detecting Security Vulnerabilities," in *Proceedings of the 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)* (IEEE, 2017), 451–455, https://doi.org/10.1109/ICRITO.2017.8342469.

4. D. Y. Perwej, S. Q. Abbas, J. P. Dixit, D. N. Akhtar, and A. K. Jaiswal, "A Systematic Literature Review on the Cyber Security," *International Journal of Scientific Research and Management* 9, no. 12 (2021): 669–710, https://doi.org/10.18535/ijsrm/v9i12.ec04.

5. A. Rastogi and K. Nygard, "Cybersecurity Practices From a Software Engineering Perspective," in *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)* (Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2017).

6. L. Futcher, "SecSDM: A Model for Integrating Security Into the Software Development Life Cycle," in *5th World Conference on Information Security Education* (Springer, 2007), 41–48, https://doi.org/10.1007/978-0-387-73269-5_6.

7. S. Kalhoro, M. Rehman, V. Ponnusamy, and F. B. Shaikh, "Extracting Key Factors of Cyber Hygiene Behaviour Among Software Engineers: A Systematic Literature Review," *IEEE Access* 9 (2021): 99339–99363, https://doi.org/10.1109/ACCESS.2021.3097144.

8. M. Tahaei and K. Vaniea, "A Survey on Developer-Centred Security," in *Proceedings of the 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)* (IEEE, 2019), 129–138, https://doi.org/10.1109/EuroSPW.2019.00021.

9. F. D. Nembhard, M. M. Carvalho, and T. C. Eskridge, "Towards the Application of Recommender Systems to Secure Coding," *EURASIP Journal on Information Security* 2019, no. 1 (2019): 9, https://doi.org/10.1186/s13635-019-0092-4.

10. R. J. Ellison, J. B. Goodenough, C. B. Weinstock, and C. Woody. "Evaluating and Mitigating Software Supply Chain Security Risks," Tech. Rep. CMU/SEI-2010-TN-016, (2010).

11. A. R. A. Hevner, S. March, et al., "Design Science in Information Systems Research," *MIS Quarterly* 28 (2004): 75–106, https://doi.org/10.2307/25148625.

12. K. Svensson, D. Axelrod, M. Mohamad, and R. Wohlrab, "Guidelines for Supporting Software Engineers in Developing Secure Web Applications," in *Product-Focused Software Process Improvement*, Vol. 15452 (Springer, 2025), 123–138, https://doi.org/10.1007/978-3-031-78386-9_9.

13. W3Tech, "Usage Statistics of Server-Side Programming Languages for Websites," (2025), https://w3techs.com/technologies/overview/programming_language.

14. B. E. Strom and A. Applebaum, *MITRE ATT&CK: Design and Philosophy*," Tech. rep. (MITRE Corporation, 2018).

15. A. Siderova, M. Daneva, F. A. Bukhsh, and J. J. Arachchige, "Security Approaches in Model-Driven Engineering for Web Applications: The State-of-the-Art in the Last 10 Years," in *2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW)* (IEEE, 2024), 155–163, https://doi.org/10.1109/REW61692.2024.00026.

16. S. Kent, "Model Driven Engineering," in *International Conference on Integrated Formal Methods*, Vol. 2335 (Springer, 2002), 286–298, https://doi.org/10.1007/3-540-47884-1_16.

17. C. Weir, I. Becker, and L. Blair, "A Passion for Security: Intervening to Help Software Developers," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)* (IEEE, 2021), 21–30, https://doi.org/10.1109/ICSE-SEIP52600.2021.00011.

18. Y. Acar, C. Stransky, D. Wermke, C. Weir, M. L. Mazurek, and S. Fahl, "Developers Need Support, Too: A Survey of Security Advice for Software Developers," *IEEE Cybersecurity Development (SecDev)* 2017 (2017): 22–26, https://doi.org/10.1109/SecDev.2017.17.

19. D. S. Battina, "Best Practices for Ensuring Security in DevOps: A Case Study Approach," *International Journal of Innovations in Engineering Research and Technology* 4, no. 11 (2017): 38–45.

20. M. A. Akbar, K. Smolander, S. Mahmood, and A. Alsanad, "Toward Successful DevSecOps in Software Development Organizations: A Decision-Making Framework," *Information and Software Technology* 147 (2022): 106894, https://doi.org/10.1016/j.infsof.2022.106894.

21. R. Tóth, T. Bisztray, and L. Erdődi, "LLMs in Web Development: Evaluating LLM-Generated PHP Code Unveiling Vulnerabilities and Limitations," in *International Conference on Computer Safety, Reliability, and Security (SAFECOMP)* (Springer, 2024), 425–437, https://doi.org/10.1007/978-3-031-68738-9_34.

22. Z. Li, S. Dutta, and M. Naik, "LLM-Assisted Static Analysis for Detecting Security Vulnerabilities," preprint, arXiv, May 28, 2025, https://arxiv.org/pdf/2405.17238.

23. J. R. Tadhani, V. Vekariya, V. Sorathiya, S. Alshathri, and W. El-Shafai, "Securing Web Applications Against XSS and SQLi Attacks Using a Novel Deep Learning Approach," *Scientific Reports* 14, no. 1 (2024): 1803, https://doi.org/10.1038/s41598-023-48845-4.

24. R. Vallabhaneni, S. Pillai, S. A. Vaddadi, S. R. Addula, and B. Ananthan, "Secured Web Application Based on CapsuleNet and OWASP in the Cloud," *Indonesian Journal of Electrical Engineering and Computer Science* 35, no. 3 (2024): 1924–1932.

25. M. Fasha, F. A. Rub, N. Matar, B. Sowan, M. Al Khaldy, and H. Barham, "Mitigating the OWASP Top 10 for Large Language Models Applications Using Intelligent Agents," in *2024 2nd International Conference on Cyber Resilience (ICCR)* (IEEE, 2024), 1–9, https://doi.org/10.1109/ICCR61006.2024.10532874.

26. S. Liu, B. Sabir, S. I. Jang, et al., "From Solitary Directives to Interactive Encouragement! LLM Secure Code Generation by Natural Language Prompting," preprint, arXiv, October 18, 2024, https://doi.org/10.48550/arXiv.2410.14321.

27. V. Yosifova, "Application of Open-Source Large Language Model (LLM) for Simulation of a Vulnerable IoT System and Cybersecurity Best Practices Assistance," preprint, Preprints, May 17, 2024, https://doi.org/10.20944/preprints202405.1169.v1.

28. A. Sajadi, B. Le, A. Nguyen, K. Damevski, and P. Chatterjee, "Do LLMs Consider Security? An Empirical Study on Responses to Programming Questions," *Empirical Software Engineering* 30, no. 3 (2025): 101, https://doi.org/10.1007/s10664-025-10658-6.

29. Aikido, "SAST by Aikido: Supported Languages and Security Focus," (2025), https://help.aikido.dev/doc/sast-by-aikido-supported-languages-and-security-focus/docsXg3y6sav.

30. Aikido, "Add Custom SAST & IaC Rules," (2025), https://help.aikido.dev/doc/add-custom-sast-rules/doce2VjBeMXk?_gl=1%2A1x3ehww%2A_gcl_au%2AMTM1NjA1MDU3MC4xNzQ3MjEwNjM3%2AFPAU%2AMTM1NjA1MDU3MC4xNzQ3MjEwNjM3.

31. Semgrep Inc., “Writing Rules,” (2025), https://semgrep.dev/docs/writing-rules/overview.

32. O. Arteau, *Prototype Pollution Attack in NodeJS Application* (NorthSec Olivier Arteau, 2018).

33. Laravel, “Laravel,” (2025), https://laravel.com.

34. S. Kirsten, “Cross Site Scripting (XSS),” (2025), https://owasp.org/www-community/attacks/xss/.

35. Team CSS, “Laravel Cheat Sheet—Cross Site Scripting (XSS),” (2025), https://cheatsheetseries.owasp.org/cheatsheets/Laravel_Cheat_Sheet.htmlcross-site-scripting-xss.

36. CURLOPT_SSL_VERIFYPEER, “Explained,” (2025), https://curl.se/libcurl/c/CURLOPT_SSL_VERIFYPEER.html.

37. Group TPD, “ftp_connect,” (2025), https://www.php.net/manual/en/function.ftp-connect.php.

38. H. Luo, “SSRF Vulnerability Attack and Prevention Based on PHP,” in *2019 International Conference on Communications, Information System and Computer Engineering (CISCE)* (IEEE, 2019), 469–472.

39. Group TPD, “fopen,” (2025), https://www.php.net/manual/en/function.fopen.php.

40. OWASP Foundation Inc., “Path Traversal,” (2025), https://owasp.org/www-community/attacks/Path_Traversal.

41. Twig, “escape,” (2025), https://twig.symfony.com/doc/3.x/filters/escape.html.

42. PortSwigger, “How to Prevent XSS,” (2025), https://portswigger.net/web-security/cross-site-scripting/preventing.

43. J. Lobacevski, “GHSL-2020-210: Template Injection in the GitHub Workflow of Hyperspacedev/Starlight Repository,” 2025, https://securitylab.github.com/advisories/GHSL-2020-210-hyperspacedev-starlight-workflow/.

44. Aysunitai, “Understanding and Preventing XSS Attacks in PHP Applications,” (2025), https://medium.com/@aysunitai/understanding-and-preventing-xss-attacks-in-php-applications-acd7d2bacb5e.

45. Group TPD, “phpinfo,” (2025), https://www.php.net/manual/en/function.phpinfo.php.

46. J. Lobacevski, “Keeping Your GitHub Actions and Workflows Secure Part 1: Preventing pwn Requests,” (2025), https://securitylab.github.com/resources/github-actions-preventing-pwn-requests/.

47. A. Nazari, “Timing Attacks in Node.js,” (2025), https://dev.to/silentwatcher_95/timing-attacks-in-nodejs-4pmb.

48. Group TPD, “Execution Operators,” (2025), https://www.php.net/manual/en/language.operators.execution.php.

49. Mongo DBI, “db.collection.findOne() (mongosh method),” (2025), https://www.mongodb.com/docs/manual/reference/method/db.collection.findOne/.

50. Mongo DBI, “Query Predicates—Alphabetical List of Operators,” (2025), https://www.mongodb.com/docs/manual/reference/mql/query-predicates/alphabetical-list-of-operators.

51. A. Ibrahim, “NoSQL Injection & Exploitation Techniques,” (2025), https://anas0x1.medium.com/whats-nosql-injection-and-how-to-exploit-it-6acad517d50e.

52. Group TPD, “var_dump,” (2025), https://www.php.net/manual/en/function.var-dump.php.

53. Mozilla Foundation, “eval,” (2025), https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval.

54. Vue.js, “Security—HTML Injection,” (2025), https://vuejs.org/guide/best-practices/securityhtml-injection.

55. Group TPD, “eval,” (2025), https://www.php.net/manual/en/function.eval.php.

56. GitHub Inc., “Secure Use Reference—Using Third-Party Actions,” (2025), https://docs.github.com/en/actions/reference/security/secure-useusing-third-party-actions.

57. Group TPD, “exec,” (2025), https://www.php.net/manual/en/function.exec.php.

58. Group TPD, “unserialize,” (2025), https://www.php.net/manual/en/function.unserialize.php.

59. Sparkle Web, “Understanding the Risks of Using Inner HTML in Web Development,” (2025), https://medium.com/@sparklewebhelp/understanding-the-risks-of-using-inner-html-in-web-development-30d4fa67f815.

60. Team CSS, “SQL Injection Prevention Cheat Sheet - Anatomy of A Typical SQL Injection Vulnerability,” (2025), https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.htmlanatomy-of-a-typical-sql-injection-vulnerability.

61. npm, “mustache.js—Logic-Less Mustache Templates With JavaScript,” (2025), https://www.npmjs.com/package/mustache.

62. Y. Katz, “Handlebars,” (2025), https://handlebarsjs.com.

63. Snyk, “Cross-Site Scripting (XSS),” (2025), https://security.snyk.io/vuln/npm:mustache:20110814.

64. D. Aleyka and P. Mishra, “Study on Manual Auditing for Web Application Vulnerability Detection,” *Annals of RSCB* 25, no. 4 (2021): 19612–19618.

65. N. Demir, T. Urban, K. Wittek, and N. Pohlmann, “Our (In)secure Web: Understanding Update Behavior of Websites and Its Impact on Security,” in *International Conference on Passive and Active Network Measurement* (Springer International Publishing, 2021), 76–92.

66. T. Rangnau, v. R. Buijtenen, F. Fransen, F. Turkmen. Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines. *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)* (IEEE, 2020), 145–154, https://doi.org/10.1109/EDOC49727.2020.00026.

67. E. A. Altulaihan, A. Alismail, and M. Frikha, “A Survey on Web Application Penetration Testing,” *Electronics* 12, no. 5 (2023): 1229.

68. T. Gasiba, U. Lechner, M. Albuquerque, and D. Fernandez, “Awareness of Secure Coding Guidelines in the Industry—A First Data Analysis,” in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)* (IEEE, 2020), 345–352, https://doi.org/10.1109/TrustCom50675.2020.00055.

69. T. Petranović and N. Zarić, “Effectiveness of Using OWASP TOP 10 as AppSec Standard,” in *2023 27th International Conference on Information Technology (IT)* (IEEE, 2023), 1–4, https://doi.org/10.1109/IT57431.2023.10078626.

70. M. Awad, M. Ali, M. Takruri, and S. Ismail, “Security Vulnerabilities Related to Web-Based Data,” *Telkomnika (Telecommunication Computing Electronics and Control)* 17 (2019): 852–856.

71. S. Myagmar, A. J. Lee, and W. Yurcik, “Threat Modeling as a Basis for Security Requirements,” in *Symposium on Requirements Engineering for Information Security (SREIS)* (University of Pittsburgh, 2005).

72. P. E. Strandberg, “Ethical Interviews in Software Engineering,” in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (IEEE, 2019), 1–11, https://doi.org/10.1109/ESEM.2019.8870192.

73. M. Chen, J. Tworek, H. Jun, et al., “Evaluating Large Language Models Trained on Code,” (2021).

74. S. Thakur, B. Ahmad, H. Pearce, et al., “VeriGen: A Large Language Model for Verilog Code Generation,” *ACM Transactions on Design Automation of Electronic Systems* 29, no. 3 (2024): 46, https://doi.org/10.1145/3643681.

75. J. Saldaña, *The Coding Manual for Qualitative Researchers* (Sage Publications, 2021).

76. M. A. Lauri, “Triangulation of Data Analysis Techniques,” *Papers on Social Representations* 20, no. 2 (2011): 34.1–34.15.

77. S. Easterbrook, J. Singer, M. A. Storey, and D. Damian, “Selecting Empirical Methods for Software Engineering Research,” in *Guide to Advanced Empirical Software Engineering* (Springer, 2008), 285–311.

78. K. R. Larsen, R. Lukyanenko, R. M. Mueller, et al., “Validity in Design Science Research,” in *Designing for Digital Transformation. Co-Creating Services With Citizens and Industry*, Vol. 12388 (Springer, 2020), https://doi.org/10.1007/978-3-030-64823-7_25.

79. T. Espinha Gasiba, K. Beckers, S. Suppan, and F. Rezabek, “On the Requirements for Serious Games Geared Towards Software Developers in the Industry,” in *2019 IEEE 27th International Requirements Engineering Conference (RE)* (IEEE, 2019), 286–296, https://doi.org/10.1109/RE.2019.00038.

80. M. Nicho, I. Effiong, and C. D. McDermott, “Shift-Left Security: Integrating Security in the Initial Phase of the DevOps Methodology,” in *2025 9th International Conference on Cryptography, Security and Privacy (CSP)* (IEEE, 2025), 182–191.

81. A. Cornelius, A. Crouch, F. Macatuno, M. Jackson, W. Johnson, and D. M. Haderlie, *Understanding How Organizations Handle Cybersecurity*.” Tech. rep. (Idaho National Laboratory (INL), 2021).

82. S. K. Lala and A. Kumar, “Secure Web Development Using OWASP Guidelines,” in *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)* (IEEE, 2021), 323–332.

83. G. Avramescu, M. Bucicoiu, D. Rosner, and N. Tăpuş, “Guidelines for Discovering and Improving Application Security,” in *2013 19th International Conference on Control Systems and Computer Science* (IEEE, 2013), 560–565.

84. S. Yang, Q. Hou, S. Li, F. Xu, and W. Diao, “From Guidelines to Practice: Assessing Android App Developer Compliance With Google's Security Recommendations,” *Empirical Software Engineering* 30, no. 1 (2025): 11.

Supporting Information

Additional supporting information can be found online in the Supporting Information section. Supplementary material LLM Experiments.zip Supplementary Material.zip.