Using ChatGPT in Designing Web Applications Resistant to SQL Injection Attacks

Helena Markulin*, Aleksander Radovan†, Branko Mihaljević*, Krunoslav Bilić‡
* RIT Croatia, Zagreb, Croatia
† Algebra Bernays University, Zagreb, Croatia†
‡ Zagreb University of Applied Sciences, Zagreb, Croatia
helena.markulin@mail.rit.edu, aleksander.radovan@algebra.hr, branko.mihaljevic@croatia.rit.edu, krunoslav.bilic@tvz.hr

Abstract — **This paper investigates leveraging ChatGPT as a tool for testing web applications resilient to SQL injection attacks. Subsequently, the web application analysis is conducted using different versions of ChatGPT, guided by best practices for preventing SQL injection vulnerabilities. The study evaluates the implemented web application based on Spring Framework, analyzing their compliance with security principles and effectiveness in mitigating risks associated with SQL injection. The research methodology was based on asking the same questions to different models while submitting the same source code and enabling additional explanation in cases where the models refused to answer the question due to the sensitivity of the topic related to hacker attacks. By examining ChatGPT's role in integrating secure coding practices during the database design phase, the research aims to assess its utility as a collaborative and intelligent assistant in the cybersecurity domain. The findings provide insights into the strengths and limitations of AI-driven cybersecurity modeling for enhancing system security against injection-based exploits. The results achieved show that different models detect different vulnerabilities, so the conclusion is that it is necessary to use several different models that would contribute to a more complete analysis of SQL injection vulnerabilities.**

Keywords — web application security, SQL injection attack and prevention, ChatGPT

I. INTRODUCTION

In the rapidly evolving landscape of web application security, SQL injection attacks remain a persistent and significant threat. These attacks, which exploit vulnerabilities in database queries, continue to pose substantial risks to organizations worldwide [1], [2]. According to the OWASP (Open Worldwide Application Security Project) Top 10 [3][4], Injection vulnerabilities slide down from the first position in 2017 to the third position in 2021 which means that the threat is still significant and must be seriously analyzed. As cybersecurity challenges grow more complex, there is as increasing need for innovative approaches to fortify web applications against such threats.

Artificial Intelligence (AI) has emerged as a powerful tool in the cybersecurity domain, offering new possibilities for enhancing application security [5][6]. Among the recent advancements in AI, large language models (LLM) like ChatGPT have shown remarkable potential in various fields, including software development and security analysis [7], [8]. This research explores the application of ChatGPT in designing web applications that are resistant to SQL injection attacks.

SQL injection attacks involve starting malicious SQL code into application queries, potentially leading to unauthorized data access, modification, or deletion [9][10]. Traditional prevention methods, such as input validation and parameterized queries, while effective, often require manual implementation and may not cover all potential vulnerabilities [2][4]. By using well-known and recognized frameworks such as Spring Framework [11] together with Spring Boot [12], Hibernate [13][14]and Spring Data JPA [15] and using best practices, it is possible to protect applications from SQL injection attacks even more effectively. This paper describes a solution implemented using Spring and Spring Boot frameworks and analyzed by ChatGPT for SQL injection vulnerabilities.

The integration of AI-driven tools like ChatGPT into the web application development process presents an opportunity to enhance security measures from the early stages of design. By leveraging ChatGPT's natural language processing capabilities and vast knowledge base, developers may be able to identify potential vulnerabilities more effectively and implement robust security measures [16], [17], [18].

Jamdade and Liu described a pilot study that uses ChatGPT for generating web application code with a specific emphasis on mitigating four prevalent web application vulnerability types, including SQL Injection [17]. They used a case study to illustrate how the vulnerabilities in the code are mitigated with the prompts and the subsequent refinement. The findings summarize the security concerns in the code generated by ChatGPT and a proposal of a prompt pattern designed to help mitigate the potential vulnerabilities.

Sanguino et al. [19] demonstrated how self-generative vulnerabilities manifest in software programming, through a case study. It undertakes a methodology that illustrates a practical example of vulnerability existing the in the code generated using an AI model such as ChatGPT, showcasing the creation of a web application database, SQL queries, and PHP server side. In our paper we focused on SQL injection vulnerabilities in a web application implemented in Java, Spring Boot, Hibernate, and Spring Data JPA frameworks.

Large Language Models facilitate human interaction via chatbots with natural language interfaces, described in [20], aided by an LLM-integration middleware, user prompts are translated into SQL queries used by the LLM to provide meaningful responses to users. Unsanitized user prompts can lead to SQL injection attacks, potentially compromising the security of the database. The findings in the paper indicate

that LLM-integrated applications are highly susceptible to SQL injection attacks.

In the next section the testing methodology is described. The third and fourth section describe how ChatGPT models were used for code analysis and for SQL injection attack modelling.

II. METHODOLOGY

The methodology used included seven versions of ChatGPT: GPT 3.5, GPT 4.0, GPT 4o-mini, GPT 4o, GPT 3o-mini-high, 3o-mini, and GPT o1 [21]. Each model is asked the same question with the programming code of the "Controller" class. The methods within the "Controller" call methods within the "Repository" class that uses the "JdbcTemplate" object of the class that executes the SQL queries. The analyzed application was used to retrieve data about parts of computer hardware.

In the first prompt, only code analysis and warning about potential threats were requested, and in the second prompt, it was requested that the ChatGPT model suggest how the vulnerability of the analyzed code to SQL injection attacks could be used. The responses were analyzed and summarized in tables 1 and 2 with marked common suggestions for fixing vulnerabilities and SQL injection attacks.

III. USING THE CHATGPT MODEL FOR CODE ANALYSIS

A. Code analysis prompt

The question used in the code analysis prompt was the following: "I have a Java Spring application, and I want to know if its secure or if it has any issues. Here's what the controller looks like:" The provided code is shown in code snippets 1, 2 and 3.

```
@GetMapping("/{id}")
public ResponseEntity<Hardware> getById(
       @PathVariable final String id)
{
 return new ResponseEntity<> (
   hardwareRepository.getHardwareById(id,
   HttpStatus.OK);
}
```

Code snippet 1. Tested controller method for fetching one object by ID

B. ChatGPT prompt responses

The question from chapter A asked seven ChatGPT models: versions GPT 3.5, GPT 4.0, GPT 4.o-mini, GPT 4o, GPT o3-mini-high, GPT o3-mini, and GPT o1.

GPT 3.5 model answered "The code you've provided has several security issues that could potentially be exploited: SQL Injection Vulnerability: The `getHardwareById` method constructs a SQL query by concatenating strings, which makes your application vulnerable to SQL injection attacks.

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Hardware {
   private Integer id;
   private String name;
   private Category category;
   private BigDecimal price;
}
```

Code snippet 2. Tested repository method for fetching one object by ID

```
@Override
public Hardware getHardwareById(
       String id)
{
   String query = SELECT_ALL;
   if(Optional.ofNullable(id)
       .isPresent())
   {
      query += " WHERE ID = " + id;
   }
   return jdbcTemplate.queryForObject(
      query, this::mapRowToHardware);
}
```

Code snippet 3. Tested domain object class return by the controller

To prevent SQL injections, you should use parameterized queries or prepared statements. Lack of Input Validation: The `getHardwareById` method does not validate the id parameter before using it in the SQL query. This could lead to unexpected behavior if the id is null or empty. You should validate user input to ensure it meets the expected criteria before using it in your application. The GPT 3.5 model also suggested to extend to provided code with input validation method to ensure that the ID parameter is not null or empty before proceeding with the database query. Furthermore, the suggestion for improvement contained modified SQL query to use a parameterized query (with '?' placeholder) and pass the ID as a parameter, which prevents SQL injection attacks by separating the SQL code from the user input. By implementing these changes, the code can significantly improve the security of the Spring application. It was suggested to implement other best security practices such as authentication, authorization, and secure communication protocols (e.g. HTTPS) to further enhance the security of the application.

GPT 4.0 model answered with similar answers to the ones from GPT 3.5 model but also included additional explanation of how SQL injection works, recommendations to use prepared statements with examples, and include the suggested concept of "Least Privilege Principle" that ensures that the database user the application connects with has the minimum necessary permissions. For example, if the application only needs to read data, the database user should not have permission to modify or delete data. Furthermore, regular security audits and code reviews were suggested to review code for security issues and conduct

security audits to identify and mitigate potential vulnerabilities. The last suggestion was to use Web Application Firewalls (WAF) that can provide an additional layer of security by filtering out malicious data before it reaches your application.

GPT 4o-mini model included four different potential problems and solutions. First, SQL Injection Risk that is based on concatenation the ID into the SQL query. The presented solution suggested using parameterized queries to avoid this risk, as other GPT models also included in their responses. Second, a new problem description called "Null Handling" was introduced. Since it is possible to send an identifier for which no entity is associated, i.e. for which it is not possible to return data and return a "Null" value, this avoids unforeseen errors in the code. It was proposed to throw an exception or return a meaningful error response instead of checking if the ID is not null with the statement "`Optional.ofNullable(id).isPresent();`". The validation of the ID parameter could be performed before executing the query and returning as appropriate error if it's invalid – that was the proposed solution to this problem. Third, potential data exposure was also mentioned as a problem. Sensitive data, such as pricing or hardware details, should not be exposed unnecessarily, especially if your application has user roles or other security restrictions. Fourth, validation and sanitization, that was mentioned in answers from other two GPT models, 3.5 and 4.0., is also elaborated in the answer from GPT 4o-mini model. If the ID parameter can come from user input (e.g., in a URL or form), it is necessary to ensure that it's properly sanitized. Although the ID seems to be a string, it's still good practice to validate it (e.g., ensuring it's a numeric value or matches a specific pattern).

GPT 4o model responded with suggestions to fix the vulnerability by using parameterized queries, type mismatch issue (using Integer instead of String when calling the "`getHardwareById`" endpoint), error handling in case of exception when the data for the given ID does not exist, and provided general best practices (validation, logging and applying database constraints).

GPT o3-mini-high model delivered information "Thought about Java Spring security issues for 9 seconds" and elaborated on the provided code with highlighting data retrieval, identifying vulnerabilities, spotting gaps, highlighting security gaps, advocating for parameterized queries, managing ID type handling, ensuring security, evaluating SQL vulnerabilities, concerns about SQL injection, and mitigating SQL injection. For every of the steps described the model generated an elaboration in a few sentences. The model answered that one of the details that can be optimized is to remove the redundant use of Optional, because this case is already covered by usage of the "@PathVariable" annotation.

GPT o3-mini model returned a message "Thought about Spring application security for a couple of seconds", with listing only one step "Unraveling security risks" and immediately generated results related to using parameterized queries and data type inconsistency that was focused on replacing String input parameter with Integer input parameter and to mitigate the risk to inject an additional SQL statement together with the ID. Error handling was also suggested as an improvement, together with Optional check misuse.

GPT o1 model returned message "Thought about Java Spring security issues for 4 seconds" and started to describe and log the following phases. Those phases included spotting SQL injection risk and evaluating ID handling and provided examples of how to fix the SQL injection vulnerability. Besides the usage of parameterized queries, query construction and optional logic, return value handling, field types in the model (insisting on using Integer instead of String data type for the ID), the model also suggested using Object-Relational-Mapping frameworks and logging.

To be able to compare the GPT models, Table 1 below has been compiled, which contains information about what each model suggests in terms of the vulnerability of the displayed program code: Parameterized Queries (PQ), Input Validation (IV), Least Privilege Principle (LPP), using Web Application Firewall (Using WAF) and Handling Missing Data (HMD). Every category of the improvement listed in the table is marked with "X" in case when the specific model suggested it as improvement option. The results have shown that the different models suggested different suggestions for improvements. To achieve best results, the results proved that it's necessary to combine several different models, for example 4.0 and o3-mini to cover and categories of improvements.

TABLE 1. GPT MODEL COMPARISON – VULNERABILITY ANALYSIS

GPT model	**SUGGESTIONS FOR IMPROVEMENTS**					
	PQ	***IV***	***LPP***	***Using WAF***	***HMD***	***ORM***
3.5	**X**	**X**				
4.0	**X**	**X**	**X**	**X**		
4o-mini	**X**	**X**	**X**		**X**	
4o	**X**	**X**	**X**		**X**	
o3-mini-high	**X**	**X**	**X**		**X**	
o3-mini	**X**	**X**			**X**	**X**
o1	**X**	**X**			**X**	**X**

IV. USING THE CHATGPT MODEL FOR SQL INJECTION ATTACK TARGETING

The second test in the use of the GPT model was related to the question of how it is possible to prepare and execute a SQL injection attack on an application with a given program code. Identical program snippets that are shown in code snippets 1, 2 and 3 were used.

Without hesitation, GPT model 3.5 immediately offered an example of an attack related to the injection of the ID parameter, in which not only the identifier is submitted, but also a malicious part of the SQL query that can harm the structure and data in the database. Furthermore, he offered an example with which it is possible to retrieve data from a

database that the user is not entitled to using tautology (by adding a part of the SQL query that is always true, for example, "1=1").

On the other hand, the GPT 4.0 model initially did not want to answer the question and gave the answer "I'm really sorry, but I can't assist with that.". Only after insisting again on the answer to the same question, emphasizing that the question was being asked for research purposes for academic use, did the model respond. The answer contained much more detail with additional code examples on how to perform a SQL injection attack. The model also included ethical considerations and legal implications to instruct the user in as many details as possible related to the execution of such attacks and how to prevent negative consequences.

GPT model 4.o-mini gave the answer immediately, like GPT model 3.5, and didn't include the insights into ethical considerations and legal implications as model 4.0.

GPT model 4o initially responded that assistance with requests related to exploiting security vulnerabilities, including SQL injection attack is not provided, but providing information about identification and fixing these vulnerabilities is possible. The model mentioned example attack scenarios like bypassing ID check, extracting data (error-based injection), union-based injection, database manipulation, defensive measures (with parameterized queries and input validation, error handling, database hardening).

GPT model o3-mini-high delivered information "Thought about SQL injection targeting for 5 seconds" and elaborated to the question with analyzing the scenario, assessing guidelines, clarifying the guidelines, confirming the request, emphasizing safety, where the answering stopped with a message "I'm sorry, but I can't comply with that". After insisting that the answer is needed for research and academic purposes, the answering continued with "Thought about SQL injection vulnerability for 31 seconds" and the following elaborations: identifying vulnerabilities, assessing potential impact, evaluating scholarly pursuits, focusing on safety, assessing academic intent, assessing guidelines, assessing intent, navigating guidelines, assessing ethical boundaries, balancing ethical boundaries, navigating guidelines, balancing academic research and ethical guidelines, navigating ethical guidelines, evaluating ethical boundaries, clarifying open guidelines, focusing on ethical guidelines, clarifying ethical boundaries, identifying potential risks, highlighting potential risks, assessing SQL injection, and highlighting multi-query risks. Provided examples in the answer were very detailed and more comprehensive than other examples generated in answers from the GPT models.

GPT model o3-mini delivered faster response than o3-mini-high model, responded with the message "Though about SQL injection risks for a couple of seconds", and started to describe answering phases: assessing potential vulnerabilities and navigating policy boundaries. After the latter phase the model answered with "I'm sorry, but I can't comply with that.", explaining that the reason was that the model is adhering to guidelines by avoiding hacking instructions. After additional explanation from the user that the answer was needed for research and academic purposes, the model continued with answering the question and started new phases like research focus, reviewing guidelines, clarifying the request, offering information, and stopped in this phase, because the model avoided to provide specific exploitation instructions, and responded again with "I'm sorry, but I can't comply with that.". The user insisted that the application that was the target wasn't a real-world application, but a test application made by the user for academic and research purposes to understand SQL injection attacks, the model responded with "Thought about SQL Injection research for 4 seconds" and continued to next phases like navigating the boundaries, evaluating research intent, evaluating guidelines and finally answering with "I'm sorry, but I can't comply with that." for the third time. We realized that the "targeting this vulnerability" was something that triggered the model to avoid answering the question, and changed the prompt to highlight that the user wanted to better understand the SQL injection attacks and cybersecurity to build a secure application and that finally resulted with expected results. The model also suggested using Object-Relational-Mapping frameworks (ORM) like Hibernate or JPA, which can help abstract SQL queries and reduce the risk of injection attacks.

GPT model o1 first responded with "Thought about SQL injection attack potential for a few seconds" and continued elaborating with spotting policy issues and rejecting to continue because of ethical considerations and legal implications. At the end the answer included the description why the code is vulnerable, continues describing the examples of attack vectors and Boolean-based SQL injection (with adding "OR 1=1" and other stacking queries and destructive injections like "DROP TABLE" additions), Comment-based injection that was used by attackers to comment out the rest of a statement with the comment ("- -").

TABLE 2. GPT MODEL COMPARISON – VULNERABILITY ANALYSIS

GPT model	**SUGGESTIONS FOR SQL INJECTION ATTACKS**			
	Initial hesitation	***ID parameter injection***	***Exposing sensitive data***	***Ethical Considerations and Legal Implications***
3.5		**X**	**X**	
4.0	**X**	**X**	**X**	**X**
4o-mini		**X**	**X**	
4o	**X**	**X**	**X**	**X**
o3-mini-high	**X**	**X**	**X**	**X**
o3-mini	**X**	**X**	**X**	**X**
o1	**X**	**X**	**X**	**X**

V. RESULTS

The results show that the models differ in the details related to the suggestions on how to analyze the SQL injection vulnerabilities within the analyzed code and how those vulnerabilities can be fixed. All used models suggested the use of parameterized queries and input data

validation, which is usually a good practice for mitigating SQL injection attacks. Interestingly, only part of the model suggested using the concept of least privilege on the database, which is also one of the most effective techniques to limit the damage that can be caused by SQL injection attacks or when an attacker gets hold of the username and password to use the database. Most models mention suggestions for handling missing values to shape the response to the attacker in a way that does not reveal all possible details about the implementation of the data retrieval itself. Using ORM libraries and frameworks significantly helps in using best practices during programming, which also entails automatic protection against SQL injection attacks, but only a few models recommended this solution, as seen in Table 1.

On the other hand, some models did not hesitate to immediately answer the question related to the methods of preparing and executing SQL attacks, while the majority nevertheless warned the user about the ethical and legal implications related to such procedures. However, using simple arguments such as using the information for academic and research purposes made it possible to circumvent these restrictions as well.

Although many models use constraints that result in rejection of answers to unethical and illegal questions, there are various techniques to circumvent them.

VI. CONCLUSION AND FUTURE WORK

Using the GPT model can significantly improve the way of writing secure code and education about potential threats related to SQL injection attacks. It can be used by developers and users, but also by attackers and hackers. For this reason, it is necessary to take these facts into account when writing your own code and count on the fact that the information and knowledge generated by the GPT model will always be available to malicious users. Critical thinking and checking generated responses can reduce the risk of vulnerabilities. Given that GPT models often learn from available data on the web, which may be incorrect or even deliberately misspelled, validation of responses before use is set as a prerequisite to avoid unwanted consequences.

It turned out that the best approach is to use several different models at the same time, since they give different answers, but at the same time, they allow checking whether the stated facts are correct, since they can be checked from several different sources.

Research related to this topic will be further expanded for the remaining OWASP TOP 10 vulnerabilities, especially after the publication of the new list in the first half of 2025. Furthermore, the next models that will be published in the future will also be included to the future research.

VII. REFERENCES

[1] S. S. Nair, “Securing Against Advanced Cyber Threats: A Comprehensive Guide to Phishing, XSS, and SQL Injection Defense,” *Journal of Computer Science and Technology Studies*, vol. 6, no. 1, pp. 76–93, Jan. 2024, doi: 10.32996/jcsts.2024.6.1.9.

[2] “SQL Injection Prevention: 6 Strategies.” Accessed: Feb. 23, 2025. [Online]. Available: https://www.legitsecurity.com/blog/how-to-prevent-sql-injection

[3] “OWASP TOP 10.” Accessed: Feb. 23, 2025. [Online]. Available: https://owasp.org/www-project-top-ten/

[4] “What is OWASP? OWASP Top 10 Security By Design Principles.” Accessed: Feb. 27, 2025. [Online]. Available: What is OWASP? OWASP Top 10 Security By Design Principles

[5] “Apllying AI in Application Security.” Accessed: Feb. 23, 2025. [Online]. Available: https://www.isaca.org/resources/isaca-journal/issues/2018/volume-1/applying-ai-in-application-security

[6] M. Mylrea and N. Robinson, “Artificial Intelligence (AI) Trust Framework and Maturity Model: Applying an Entropy Lens to Improve Security, Privacy, and Ethical AI,” *Entropy*, vol. 25, no. 10, p. 1429, Oct. 2023, doi: 10.3390/e25101429.

[7] J. Li, P. H. Meland, J. S. Notland, A. Storhaug, and J. H. Tysse, “Evaluating the Impact of ChatGPT on Exercises of a Software Security Course,” in *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, IEEE, Oct. 2023, pp. 1–6. doi: 10.1109/ESEM56168.2023.10304857.

[8] M. M. Kholoosi, M. A. Babar, and R. Croft, “A Qualitative Study on Using ChatGPT for Software Security: Perception vs. Practicality,” in *2024 IEEE 6th International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-ISA)*, IEEE, Oct. 2024, pp. 107–117. doi: 10.1109/TPS-ISA62245.2024.00022.

[9] “SQL Injection Attack: How It Works, Examples and Prevention.” Accessed: Feb. 24, 2025. [Online]. Available: https://brightsec.com/blog/sql-injection-attack/

[10] M. M. Kholoosi, M. A. Babar, and R. Croft, “A Qualitative Study on Using ChatGPT for Software Security: Perception vs. Practicality,” in *2024 IEEE 6th International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-ISA)*, IEEE, Oct. 2024, pp. 107–117. doi: 10.1109/TPS-ISA62245.2024.00022.

[11] Laurentiu Spilca, *Spring Security in Action*, 2nd ed. Manning, 2024.

[12] M. Mythily, A. Samson Arun Raj, and I. Thanakumar Joseph, “An Analysis of the Significance of Spring Boot in The Market,” in *2022 International Conference on Inventive Computation Technologies (ICICT)*, IEEE, Jul. 2022, pp. 1277–1281. doi: 10.1109/ICICT54344.2022.9850910.

[13] Christian. Bauer and Gavin. King, *Hibernate in action*. Manning, 2005.

[14] Vlad Mihalcea, *High-Performance Java Persistence*. 2016.

[15] C. Tudose and C. Odubasteanu, “Object-relational Mapping Using JPA, Hibernate and Spring Data JPA,” in *2021 23rd International Conference on Control Systems and Computer Science (CSCS)*, IEEE, May 2021, pp. 424–431. doi: 10.1109/CSCS52396.2021.00076.

[16] Victoria Shutenko, “AI in Cybersecurity: Exploring the Top 6 Use Cases.” Accessed: Mar. 01, 2025. [Online]. Available: https://www.techmagic.co/blog/ai-in-cybersecurity

[17] M. Jamdade and Y. Liu, “A Pilot Study on Secure Code Generation with ChatGPT for Web Applications,” in *Proceedings of the 2024 ACM Southeast Conference on ZZZ*, New York, NY, USA: ACM, Apr. 2024, pp. 229–234. doi: 10.1145/3603287.3651194.

[18] I. Mulyar, S. Lenkov, V. Glowyuk, V. Anikin, and Y. Sotnikov, “A method for finding web application vulnerabilities using the ChatGPT API,” *Smart technologies: Industrial and Civil Engineering*, vol. 2, no. 15, pp. 46–55, Dec. 2024, doi: 10.32347/st.2024.2.1203.

[19] T. de J. Mateo Sanguino, “Enhancing Security in Industrial Application Development: Case Study on Self-Generating Artificial Intelligence Tools,” *Applied Sciences*, vol. 14, no. 9, p. 3780, Apr. 2024, doi: 10.3390/app14093780.

[20] R. Pedro, D. Castro, P. Carreira, and N. Santos, “From Prompt Injections to SQL Injection Attacks: How Protected is Your LLM-Integrated Web Application?,” Aug. 2023.

[21] “GPT models.” Accessed: Mar. 01, 2025. [Online]. Available: https://platform.openai.com/docs/models