I'm Sorry Dave, I'm Afraid I Can't Fix Your Code: On ChatGPT, CyberSecurity, and Secure Coding

Tiago Espinha Gasiba
Siemens AG, München, Germany

Kaan Oguzhan
Siemens AG, München, Germany

Ibrahim Kessba
Siemens AG, München, Germany

Ulrike Lechner
Universität der Bundeswehr München, Germany

Maria Pinto-Albuquerque
Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR, Portugal

Abstract

Software security is an important topic that is gaining more and more attention due to the rising number of publicly known cybersecurity incidents. Previous research has shown that one way to address software security is by means of a serious game, the CyberSecurity Challenges, which are designed to raise awareness of software developers of secure coding guidelines. This game, which has been proven to be very successful in the industry, makes use of an artificial intelligence technique (laddering technique) to implement a chatbot for human-machine interaction.

Recent advances in machine learning led to a breakthrough, with the implementation of ChatGPT by OpenAI. This algorithm has been trained in a large amount of data and is capable of analysing and interpreting not only natural language, but also small code snippets containing source code in different programming languages. With the advent of ChatGPT, and previous state-of-the-art research in secure software development, a natural question arises: *to which extent can ChatGPT aid software developers in writing secure software?*.

In this paper, we draw on our experience in the industry, and also on extensive previous work to analyse and reflect on how to use ChatGPT to aid secure software development. Towards this, we run a small experiment using five different vulnerable code snippets. Our interactions with ChatGPT allow us to conclude on advantages, disadvantages and limitations of the usage of this new technology.

2012 ACM Subject Classification Applied computing → Learning management systems; Security and privacy → Software security engineering; Applied computing → Distance learning; Applied computing → E-learning

Keywords and phrases Serious Games, IT-Security, Machine Learning, ChatGPT, Secure Coding, Industry, Software Development, Teaching

Digital Object Identifier 10.4230/OASIcs.ICPEC.2023.2

Funding *Ulrike Lechner*: Acknowledged funding for project LIONS by dtec.bw.
Maria Pinto-Albuquerque: This work is partially financed by national funds through FCT - Fundação para a Ciência e Tecnologia, I.P., under the projects FCT UIDB/04466/2020 and UIDP/04466/2020. Furthermore, Maria Pinto-Albuquerque acknowledges and thanks the Instituto Universitário de Lisboa and ISTAR, for their support.

1 Introduction

According to ISO 25000 [16], one aspect of software development is security. Software security has been gaining much attention over the last decade due to the increasing number of cybersecurity incidents that are caused by poor software development practices. As a consequence, industrial standards such as IEC 62.443 mandate the implementation of a secure software development life cycle, to address and lower the number of vulnerabilities in products and services. The development of secure software is not only an important topic for the industry (e.g. in critical infrastructures), but it is also an important subject taught in may engineering and informatics courses at several universities.

There are several known methods to improve the quality of software. Among others, some of these methods include: performing secure code reviews, usage of static application security testing (SAST), and employment of security testing techniques such as unit testing, penetration testing, and fuzzing. These methods to improve software security are generally based on the fact that software should comply to a set of secure coding guidelines; secure coding guidelines are policies aimed at minimizing vulnerabilities and bugs in software. One way to ensure that software follows secure coding guidelines is by means of the usage of static application security testing tools. However, not all secure coding guidelines are decidable [3], i.e. there exist some secure coding guidelines for which no theoretical Turing Machine (TM) can be constructed such that, given some source code the TM identifies compliance or non compliance to the guideline. As a result from this theoretical perspective, an immediate problem is raised: full automation of secure coding is not possible. Software developers are ultimately responsible for the security of the code they write. However, in a 2019 survey with more than 4000 software developers from the industry, Patel [21] has shown that more than 50% of them cannot recognize vulnerabilities in source code.

A way to address this problem is to raise awareness of secure coding among software developers. Similar to Patel, in [11], Gasiba has shown that industrial software developers' lack awareness of secure coding guidelines. He extended the work by Hänsch et al. to the field of secure coding, defining secure coding awareness in three dimensions: perception, protection, and behavior.

The recent advances in technology, in particular in Machine Learning (ML), allow new techniques to be used to assist software developers to write secure code. In [8], Gasiba et al. have shown that artificial intelligence can to be used to raise awareness of software developers. The authors devised an intelligent coach by means of an artificial intelligent technique - the laddering technique - which is mostly used in chatbots [22]. The intelligent coach, which allowed Human-Machine interaction (HMi) in a controlled environment (the Sifu platform), was shown to be very successful to raise awareness of secure coding guidelines of software developers in the industry.

In this paper, we extend previous work by exploring the usage of ChatGPT [19] as a means of HMi. ChatGPT, which was released in November 2022, is built on top of the GPT-3 family of large language models, and was developed by the American research laboratory OpenAI. The language model has been fine-tuned with both supervised and reinforcement learning techniques.

Given the authors' experience, previous work, as also the theoretical limitations inherent to the secure coding field, this work aims to broaden the understanding on the extent ChatGPT can aid software developers to write secure code. This work seeks to understand *to which extent ChatGPT can recognize vulnerabilities in source code*, and *to which extent ChatGPT can rewrite code to eliminate the present security vulnerabilities.* The reason the authors have chosen ChatGPT for experimentation, as opposed to other existing generative models (in particular models trained for cybersecurity), is the fact that, to the best of our knowledge, not only is ChatGPT available to the wide public, but also allows to maintain a conversation while remembering previous requests and answers. Theregore, we achieve the present study by means of interactions with ChatGPT based on five exercises taken from the serious game CyberSecurity Challenges (CSC), and analysis of the answers in terms of secure coding. In the present work, we also reflect on the usage of ChatGPT and similar technologies as a means to teach software developers to write secure code, both in academia and also in the industry.

This work provides a valuable insight to both industry practitioners, but also to researchers, by giving an overview of the advantages, disadvantages but also limitations of using Human-Machine interactions as a means to raise awareness of secure coding. This paper also opens the doors and gives a first step towards a new and rich field of research: using Machine Learning algorithms and generative AI to raise awareness of secure coding by means of HMi.

In Section 2 we present related work that was used as basis for our research. In Section 3 we describe the setup and experiment that we used to address our research question. Section 4 presents the result as the outcome of our experiments. A discussion and reflection on the obtained results is provided in Section 5. Finally, Section 6 concludes the paper and gives an outline of future work.

2 Related Work

Several previous work was used as a basis for the current publication. Industrial security standards such as ISO/IEC 62.443 [15] motivate the work. In particular, the 4.1 part of the standard describes the implementation of processes to address the life cycle of secure software development. One important aspect that the standard mandates is the establishment of several secure coding practices, e.g. the implementation of secure coding guidelines during the development of software. Some influential secure coding guidelines are provided by the Open Web Application Security Project (OWASP) in the form of Top-10 rules [18], and the secure coding guidelines provided by the Software Engineering Institute of the Carnegie Mellon University [5]. A further cybersecurity standard widely used in the industry is given by the MITRE corporation in form of Common Weakness Enumeration [6].

One way to address IT security is given by the German BSI Grundschutzkatalog [4] standard, which recognizes serious games as a means to raise awareness of IT security. A serious game, as defined by Dörner et al. [7], is a game that is developed with a purpose that is not only entertainment. Gasiba et al. have developed a game with the purpose to raise awareness of software developers of secure coding guidelines. This game (the CyberSecurity Challenges) is based on a platform which the authors called Sifu [8, 11], and has been shown to motivate software developers to think about security. In their game, the player is presented with a secure coding challenge containing software vulnerabilities. The player interacts with an intelligent coach, which is a software component that implements an artificial intelligence (AI) engine, in order to solve the challenge. The goal of the intelligent coach is to provide hints to the players on the reasons why software is not compliant to secure coding rules. The game is played through several interactions with the intelligent coach, until a solution to the given challenge is considered acceptable by the AI algorithm. The criteria for an acceptable solution includes: (1) initial vulnerability present in the challenge is removed, (2) no additional vulnerabilities are introduced, and (3) the code respects the desired semantics, i.e. behaves as expected. The AI mechanism implemented in the Sifu plaform makes use of the chatbot laddering technique [22]. The hints that are provided to the player are thus given in an increasing level of clarity and exactness.

Although there exist other ways to increase software security, e.g. by means of static application security testing tools, our work focuses on the human factor. We motivate our choice by the fact that secure coding is a topic that cannot be fully solved by means of automation [13, 20, 2]. In [1], Acar et al. discuss how software developers search for advice for their coding activities, and conclude that software developers need assistance to understand this advice. Additionally, in 2019, Patel [21] conducted a large-scale study with over 4000 software developers. One of the results of his survey shows that more than half of software developers cannot recognize vulnerabilities in source code. Similar results were obtained by Gasiba et al. [9, 10].

Recent work has been published on the usage of machine learning algorithm to detect security vulnerabilities in source code. Harrer et al. [14] studied two feature extraction methods for C/C++ code and use this to build a control flow graph and to determine if the code contains vulnerabilities or not, without classifying them. In [23], Tang et al. extend previous work by not only looking for the presence of vulnerabilities, but also in classifying them according to MITRE's Common Weakness Enumeration (CWE) [6]. While previous approaches dealt with the C and C++ programming languages, Louati et al. extended this work for the C# programming language in [17]. To the best of our knowledge, all previous work show good indicators that machine learning is adequate to detect and classify vulnerabilities in software.

One machine learning algorithm that is currently raising lots of interest in the research community is the ChatGPT. ChatGPT is a language model developed by OpenAI that uses machine learning to generate human-like text. It is trained on a large dataset of text from the internet and is capable of understanding and responding to natural language inputs. It can be used for a wide range of tasks such as language translation, question answering, and text completion.

Not only is the algorithm based on machine learning, but its implementation allows a natural dialog between man and machine. ChatGPT processes queries from users, which are written in English, and compute an answer in an conversational way. ChatGPT not only processes the answer based on the current query, but also based on previous queries. Although it is mostly trained for natural languages, it can also interpret programming languages, such as C and C++. In the present work, we use this feature to conduct an interactive dialog with the algorithm based on five challenges from the CSC game.

3 Experiment

To setup our experiment, we selected five different challenges from the CyberSecurity Challenges, based on C/C++, and contained in the Sifu platform. Table 1 shows a summary of the selected challenges, and the corresponding CWE identifier. These challenges were chosen based both on the prevalence of the programming errors, but also based on practical experience in teaching cybersecurity from the authors'. The authors' used the 2023 January 13 version of ChatGPT, which is based on the GPT-3 training data.

The first challenge contains code of a C function that has a standard buffer overflow vulnerability. The buffer overflow in this challenge is evident through the usage of the *strcpy* function. The function in the second challenge, which is developed in C++, contains vulnerability based on undefined behavior. Depending on the compiler, the implemented function can produce different results. The third challenge makes use of a vulnerable C function, the *gets* function. Due to the problems that this function can cause, it has been deprecated and removed in the C11 standard. The fourth challenge corresponds to code that

Table 1 Selected Challenges from Sifu Platform, According to CWE ID.

ID	Vulnerability	Description
1	CWE-121	Stack-Based Buffer Overflow
2	CWE-758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior
3	CWE-242	Use of Inherently Dangerous Function
4	CWE-190	Integer Overflow or Wraparound
5	CWE-208	Observable Timing Discrepancy

contains an integer overflow vulnerability. The integer overflow can be triggered by calling the function with large integer values. Finally, the fifth and last chosen challenge contains a side-channel leakage vulnerability. The information leakage occurs due to the fact that the function performs string string comparison and the running time epends on its inputs. The last chosen challenge is more typical in embedded systems.

Listing 1 Vulnerable Code Snippet Containing CWE-208.

```
int is_equal(const char* a, const char* b, size_t len) {
    for (size_t i = 0; i < len ; i++) {
        if (a[i] != b[i])
            return 1;
    }
    return 0;
}
```

Listing 1 shows the source code corresponding to the fifth challenge. The problem with the code is that, the for loop will break depending on the contents of the input *a* and input *b*. Whenever the first difference is found, the for loop breaks and the function results 1. If both vectors contain the same values, the for loop will take the longest time to run, dependent on the length of the vectors. In the Sifu platform, the user is also given the information that the input *a* and *b* are of the same length, and that this length is equal to *len*.

For this example, the desired answer from the player corresponds to the code shown in listing 2. In this listing, the function does not return immediately when the first unequal values are observed. The run time of the function will be constant, and only dependent on the length of the vectors. Since the returned value of the comparison does not depend on the contents of the input vectors *a* and *b* but only on their length, no information is leaked by running the algorithms, i.e. an attacker able to manipulate one of the inputs cannot gain information about the other input by means of the time the algorithm takes to run.

Listing 2 Desired Challenge Solution.

```
int is_equal(const char* a, const char* b, size_t len) {
    if (a == NULL || b == NULL) return -1;

    int result = 0;
    for (size_t i = 0; i < len; i++) {
        result |= a[i] ^ b[i];
    }
    return result;
}
```

Figure 1 shows how the interaction was carried out with ChatGPT. For each challenge, a different code snippet S containing vulnerable code is provided. To test the algorithm, the author conducted several human interactions with the ML algorithm, for each code snippet S, corresponding to the five chosen challenges.

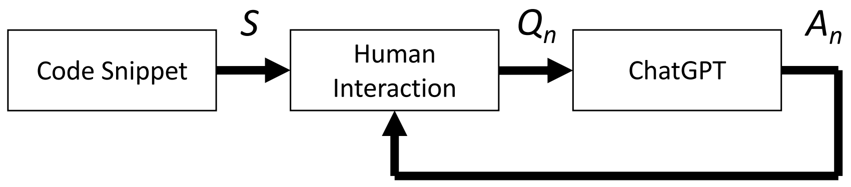

Figure 1 Interaction with ChatGPT.

The interactions with ChatGPT consisted of a separate session of questions Q_n posed to the algorithm and its corresponding answers A_n.

Table 2 Human Interactions with ChatGPT.

Nr	Question	Expected Answer
1	What is the vulnerability present in the following code snippet S	Correct vulnerability identification
2	What is the corresponding CWE number?	CWE number according to table 1
3	Please fix the code	Correct fix of the code
4..15	There is still a vulnerability in the code, please fix it	Improved code
>15	The code contains vulnerability XXX, please fix it	Improved code

The strategy to ask questions was the following. In the first question, we ask ChatGPT to identify the vulnerability by name. Since ChatGPT is verbose, we expected it to output a description of the problem. In the second question, we wanted to get the CWE number corresponding to the vulnerability to test if it matches our design. The design of both of these questions has the goal to identify how ChatGPT can support a software developer in finding and understanding secure code problems.

In the next phase (i.e. starting with question 3), we asked ChatGPT to fix the code, based on its previous answers. Our expectation is that the fixed code will correctly address the challenge vulnerability. We also carried out additional questions (4..14), where we claimed to ChatGPT that there were additional vulnerabilities and that ChatGPT should fix them. The goal of this last question was to determine how far could the algorithm could detect further problems and iteratively improve the code. Finally, on the sixteenth question, we claimed to ChatGPT that the code contained the intended vulnerability, and that ChatGPT should fix it.

The experiments were carried out through the online interface of ChatGPT on the 15th January 2023. It consisted of a total of 43 interactions with the ChatGPT user interface, corresponding to 5 for CWE-121, 5 for CWE-758, 5 for CWE 242, 11 for CWE-190, and 17 for CWE-208. The version of ChatGPT reported in the user interface was "ChatGPT 9 Jan Version".

4 Resuls

Table 3 shows a summary of the challenges and their corresponding identified vulnerability by ChatGPT. Since the CWE identified by ChatGPT was not matching exactly the CWE from the challenge, we decided to compare the solution based on the proximity of the answer from ChatGPT. We concluded that, for the first three challenges, the vulnerabilities that were identified were corresponding to a specialization of the problem. While the challenge CWE considered the general case, ChatGPT was more precise in its findings. Therefore, we concluded that the answer from ChatGPT was acceptable in those circumstances.

Table 3 Vulnerabilities Identified by ChatGPT.

ID	Designed Challenge Vulnerability	ChatGPT Identified Vulnerability	Assessment is Acceptable?	Description
1	CWE-121	CWE-120	YES	Correctly identified that buffer overflow occurs due to copying string
2	CWE-758	CWE-835	YES	Correctly identified an infinite loop due to undefined behavior
3	CWE-242	CWE-121	*YES*	Technically correct assessment, but failed to identify that gets is deprecated
4	CWE-190	CWE-835	NO	Assessment of problem misses the major issue, in favor of smaller detail
5	CWE-208	CWE-121	*NO*	While technically correct, the assessment is wrong due to missing context

For the fourth and fifth challenge (CWE-190 and CWE-121), ChatGPT did not correctly identify the problem with the code. For the CWE-190 challenge, ChatGPT found another problem, which although correct, was not the major problem with the code. Since the major problem with the code was not identified, we decided not to accept ChatGPT's solution. For the CWE-208 challenge, ChatGPT completely missed the problem. Even though we asked several times (4..15) to find and fix vulnerabilities in the code, ChatGPT could not identify it. In the 16th iteration, we told ChatGPT that the code had the CWE-208 vulnerability and that ChatGPT should solve fix the code.

Due to the interesting nature of this special case, we present the solution from ChatGPT at question 16. Listing 3 shows the solution presented by ChatGPT. In this case, ChatGPT used the CRYPTO_memcmp from OpenSSL to solve the problem

Listing 3 ChatGPT Solution to CWE-208 Challenge (using OpenSSL).

```c
#include <openssl/crypto.h>

int is_equal(const char* a, const char* b, size_t len) {
    if(a == NULL || b == NULL) return -1;
    size_t a_len = strnlen(a, len);
    size_t b_len = strnlen(b, len);
    if (len == 0 || len > a_len || len > b_len) {
        len = a_len < b_len ? a_len : b_len;
    }
    if (a[a_len] != '\0' || b[b_len] != '\0') {
        return -1;
    }
    return CRYPTO_memcmp(a, b, len);
}
```

Since we were not expecting that ChatGPT would use an external library, on iteration 17 we asked ChatGPT to provide us with a solution without using OpenSSL. The answer from ChatGPT is presented in Listing 4.

Listing 4 ChatGPT Solution to CWE-208 Challenge (without using OpenSSL).

```c
int is_equal(const char* a, const char* b, size_t len) {
    if(a == NULL || b == NULL) return -1;
    size_t a_len = strnlen(a, len);
    size_t b_len = strnlen(b, len);
    if (len == 0 || len > a_len || len > b_len) {
        len = a_len < b_len ? a_len : b_len;
    }
    if (a[a_len] != '\0' || b[b_len] != '\0') {
        return -1;
    }
    int result = 0;
    for(size_t i = 0; i < len; i++) {
        result |= a[i] ^ b[i];
    }
    return result;
}
```

In this listing, we can see that the desired solution (see listing 2) is present in the code. However, additional tests are performed in relation to the length of the input arrays, which are unnecessary, given the context in which the function is used. Furthermore, ChatGPT has assumed that both inputs represent a string and therefore tests for the case that the inputs are null-terminated. We note that these modifications to the source code introduced by ChatGPT change the semantic of the function.

5 Discussion

According to our experience in teaching secure coding in the industry, and as a result of the interactions taken with ChatGPT, we can conclude that this ML algorithm has a very high potential to be used to assist software developers in writing secure code. In more than 60% of the code that we provided to ChatGPT, it was able to correctly identify the problem with the source code, and to provide a good fix., This was very surprising since, as briefly discussed in the introduction, the problem of secure coding leads to non-decidable problems. We hypothesize that the reason for this success has to do with the fact that the code snippets that were supplied to ChatGPT were relatively small (i.e. less than 40 lines of code). Further investigations would be needed on the efficiency of ChatGPT in identifying software vulnerabilities in large code basis.

Another surprise was the fact that the explanations about the problems contained in the challenges was matching very well with the actual problem. ChatGPT's explanation was not only 3/5 of the time correct, but it was also precise in the identification and explanation of the problem. We see this as a clear advantage for ChatGPT as a teaching tool. Nevertheless, due to the fact that only a small number of snippets were tried, and that the code snippets were small in size, more investigation needs to be carried out to fully understand the usage of ChatGPT for teaching purposes.

Another point that surprised us was the fact that, not only could ChatGPT interpret the code, but could also suggest fixes to it. In particular, some of the code fixes suggested by ChatGPT could be considered to be creative. This is a clear indicator of how advanced the implementation of the algorithm is.

However, several limiting factors have also been found while interacting with ChatGPT.

We were surprised of how good the model is, but we also found limitations to its use, as it is lacking on some aspects. In the following we summarize the major aspects that we found that can limit the usage of this technology:

Missing Context. ChatGPT lacks the context in which the code is being used. This can lead to superfluous corrections and bug fixes which are not necessary due to the boundary conditions

Change Semantics. One major problem that was identified in the fourth challenge was that the solution given by ChatGPT changed the semantic of the code in a very subtle way; this means that code before fixing and after fixing can behave slightly different. This can be a strong deterrent factor to use this technology in practice (e.g. for safety-critical systems). Changes in the code should be semantic-preserving and ChatGPT currently does not guarantee this

Code Complexity. Code produced by ChatGPT has the potential to have more computational complexity than the original (vulnerable) code. This can potentially introduce computational inefficiencies, which is a critical aspect for real-time systems

Code Maintainability. Code produced by ChatGPT lacks maintainability characteristics, e.g. due to increased complexity or missing comments

Limited Learning. ChatGPT has been tuned with data up to 2021. This means that potential new threats and vulnerabilities that have been found since then might not be well processed by the system. Further investigations would need to be carried out to test the algorithm in this circumstances

Learning Interference. ChatGPT can learn from user interactions. For the algorithm to be used in a professional environment (especially in safety-critical systems and critical infrastructures), some protections need to be added such that the algorithm behind ChatGPT cannot learn incorrect data and therefore does not give bad answers.

Combining our experience, previous research , and our experiment with ChatGPT, we conclude with a reinforcement of the conclusions done in [8], and in [11]. In particular, we reinforce the conclusion that using an AI/ML engine can be an excellent approach for teaching and raising awareness of secure coding in software. Further research could integrate ChatGPT into the Sifu platform to further validate the approach with real-world scenarios and software developers in the field.

This work shows the potential that ChatGPT has to be used not only as a teaching tool, but also as a tool to assist professional software developers in the industry. We think that the tool can can assist software developers to think outside the box and find creative new solutions to complex problems. One example of the usage of generative AI technology to assist software developers write code is GitHub's Copilot [12]. However, as per conclusions on the present work, while Copilot can help software developers write code faster, further investigation is needed to understand the extent to which this software development model can introduce or eliminate the introduction of vulnerabilities in software.

Additionally, careful reflection and care must be carried out when using AI models in an industrial environment, since the model can potentially learn also from input which is provided to it. This could potentially lead to serious leakages of information to the wide public, e.g. on software weaknesses in the products and services offered by the company. Therefore, according to our experience, we consider in-house usage of AI for assisting software development to be the appropriate means to use the technology in an industrial context.

Finally, we would like to reflect on the possible usage of ChatGPT as a means to understand the output of SAST tools. Our experience has shown that software developers do not always understand the output that is provided by SAST tools and, therefore, cannot recognize the corresponding vulnerability in software. We think that ChatGPT could be used to analyse the result of these tools and to aid software developers to understand this output and therefore to write better code. However, further research is needed to validate this point.

In conclusion, we would like to highlight a further possible problem which is related to software plagiarism. ChatGPT learns from many different sources (containing many different types of software licenses) and summarizes the output of these while processing the answers to the questions. One potential problem and danger of using ChatGPT is the fact that, since it can produce code that is equal to some random source, or may be considered derivative work, this could lead to potential legal problems.

Our experience and previous research has shown that AI-based technologies can be used to effectively raise awareness of software developers, both in an industrial setting and also in the academia. Nevertheless, while we observe good indicators for ChatGPT as tool to teach cybersecurity, we are not entirely convinced that this is the case, especially for the industry, as not only the answers provided by the model can be partially wrong, it is not clear the status of the copyright infringement of the provided solutions.

To wrap-up our final conclusions, we add a text generated by ChatGPT itself:

ChatGPT is not able to fix code or help developers write secure code. While it is able to understand and respond to natural language inputs related to coding and cybersecurity, it does not have the capability to understand or execute code. It can provide general information and suggestions based on the information it has been trained on, but it cannot guarantee that the suggestions will be accurate or complete. Additionally, ChatGPT is not able to identify or fix specific vulnerabilities in code.

While ChatGPT can be a useful tool for developers, it should not be relied upon as a sole solution for ensuring the security of code. Developers should use a combination of techniques, such as threat modeling, penetration testing, and code review, to identify and fix vulnerabilities in their code. Additionally, it is important for developers to stay informed about the latest threats and vulnerabilities, and to follow secure coding practices.

5.1 Threats to Validity

The present work is based on the ChatGPT version from January 13 2023, with the training model GPT-3. As the model is being rapidly improved, part of our results might not be reflected in later versions of the improved model. This can potentially limit or invalidate part of the conclusions in the present work. While the authors' believe that the generative AI technology will experience a significant improvement, leading to potential better results as those hereby presented, we also claim that there are fundamental theoretical limitations that will certainly impose limits to its usefulness and practical applicability.

6 Conclusion

Software security is not only an important topic of software development, it has been gaining much attention over the last years. The reason for this, is the fact that there is an increasing number of cybersecurity incidents taking place that have negative consequences for society in general. A possible root-cause of cybersecurity incidents is related with poor coding practices. Therefore, to reduce the number of incidents, software developers should know and employ best practices while developing software. These practices are generally taught either in the university, or during an internal training in the industry.

In this paper, we look at the potential of using Machine Learning algorithms to both assist software developers to write secure code, but also as a tool to raise awareness of secure coding. Previous work has shown that artificial intelligence techniques can be successfully used to train software developers in secure coding guidelines. This paper presents a preliminary exploration of the usage of ChatGPT to raise secure coding awareness. Our work follows not only from the experience of the authors, but also on their extensive work in the field. In this paper, we reflect on the advantages and disadvantages of the usability of ChatGPT or similar algorithms and show that, while ChatGPT has a clear potential to be used as an aid to software development, there are some limitations to its usage. In further work, the authors would like to integrate ChatGPT with CyberSecurity Challenges – a serious game to raise awareness of secure coding guidelines of software developers in the industry. Our preliminary research presented in the present work lead us to expect good results of this integration.

References

1 Yasemin Acar, Christian Stransky, Dominik Wermke, Charles Weir, Michelle Mazurek, and Sascha Fahl. Developers Need Support, Too: A Survey of Security Advice for Software Developers. *2017 IEEE Cybersecurity Development (SecDev)*, pages 22–26, September 2017. IEEE Computer Science, Engineering. `doi:10.1109/SecDev.2017.17`.

2 Bushra Aloraini, Meiyappan Nagappan, Daniel German, Shinpei Hayashi, and Yoshiki Higo. An Empirical Study of Security Warnings From Static Application Security Testing Tools. *Journal of Systems and Software*, 110427(158):1–25, December 2019. Elsevier, Amsterdam, Nederland. `doi:10.1016/j.jss.2019.110427`.

3 Roberto Bagnara, Abramo Bagnara, and Patricia M Hill. Coding guidelines and undecidability. *arXiv preprint*, 2022. `arXiv:2212.13933`.

4 Bundesamt für Sicherheit in der Informationstechnik. BSI IT-Grundschutz-Katalog. Technical report, Bundesamt für Sicherheit in der Informationstechnik, Reguvis Fachmedien GmbH, Köln, Germany, April 2016. 15. ed, BSI. URL: `https://download.gsb.bund.de/BSI/ITGSK/IT-Grundschutz-Kataloge_2016_EL15_DE.pdf`.

5 Carnegie Mellon University. Secure Coding Standards. Software Engineering Institute, Online, Accessed 19 March 2019. URL: `https://wiki.sei.cmu.edu/confluence/display/seccode`.

6 MITRE Corporation. Common Weakness Enumeration. Online, Accessed 4 July 2019. URL: `https://cwe.mitre.org/`.

7 Ralf Dörner, Stefan Göbel, Wolfgang Effelsberg, and Josef Wiemeyer. *Serious Games: Foundations, Concepts and Practice*. Springer International Publishing, 1 edition, September 2016.

8 Tiago Gasiba, Ulrike Lechner, and Maria Pinto-Albuquerque. Sifu - A CyberSecurity Awareness Platform with Challenge Assessment and Intelligent Coach. *Special Issue of Cyber-Physical System Security of the Cybersecurity Journal*, pages 1–23, October 2020. SpringerOpen, Online. `doi:10.1186/s42400-020-00064-4`.

9 Tiago Gasiba, Ulrike Lechner, Maria Pinto-Albuquerque, and Daniel Mendez Fernandez. Awareness of Secure Coding Guidelines in the Industry - A First Data Analysis. In Guojun Wang, Ryan Ko, Md Zakirul Alam Bhuiyan, and Yi Pan, editors, *TrustCom 2020: International Conference on Trust, Security and Privacy in Computing and Communications*, pages 345–352, December 2020. IEEE, Guangzhou, China. `doi:10.1109/TrustCom50675.2020.00055`.

10 Tiago Gasiba, Ulrike Lechner, Maria Pinto-Albuquerque, and Daniel Mendez. Is Secure Coding Education in the Industry Needed? An Investigation Through a Large Scale Survey. In Hakan Erdogmus and Ana María Moreno, editors, *43rd International Conference on Software Engineering*, pages 1–12, May 2021. . URL: `https://arxiv.org/abs/2102.05343`.

11 Tiago Espinha Gasiba. *Raising Awareness on Secure Coding in the Industry through CyberSecurity Challenges*. PhD thesis, Universität der Bundeswehr München, 2021. URN: urn:nbn:de:bvb:706-7860. URL: `https://athene-forschung.unibw.de/85049?query=gasiba&show_id=140142`.

12 GitHub. Copilot. Online, Accessed 15 June 2023. URL: `https://github.com/features/copilot`.

13 Katerina Goseva-Popstojanova and Andrei Perhinschi. On the Capability of Static Code Analysis to Detect Security Vulnerabilities. *Information and Software Technology*, 68:18–33, December 2015. Butterworth-Heinemann, Newton, MA, USA. `doi:10.1016/j.infsof.2015.08.002`.

14 Jacob A. Harer, Louis Y. Kim, Rebecca L. Russell, Onur Ozdemir, Leonard R. Kosta, Akshay Rangamani, Lei H. Hamilton, Gabriel I. Centeno, Jonathan R. Key, Paul M. Ellingwood, Marc W. McConley, Jeffrey M. Opper, Sang Peter Chin, and Tomo Lazovich. Automated Software Vulnerability Detection with Machine Learning. *CoRR*, abs/1803.04497, 2018. `arXiv:1803.04497`.

15 International Electrotechnical Commission. IEC 62443-4-1 – Security for industrial automation and control systems - Part 4-1: Secure product development lifecycle requirements. Technical report, International Electrotechnical Commission, Geneval Switzerland, January 2018. .

16 International Organization for Standardization. ISO/IEC 25000:2014 – Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE. Technical report, International Organization for Standardization, Geneva, CH, March 2014. Software and Systems Engineering. URL: http://iso25000.com/index.php/en/iso-25000-standards.

17 Akram Louati and Tiago Gasiba. Source Code Vulnerability Detection using Deep Learning Algorithms for Industrial Applications. In *The Second International Conference on Ubiquitous Security (UbiSec 2022)*, pages 1–19, December 2022. .

18 Open Web Application Security Project. OWASP Top 10. Online, Accessed 15 July 2017. URL: https://tinyurl.com/yyb8wcv9.

19 OpenAI LP. ChatGPT. Online, Accessed 23 January 2023. URL: https://chat.openai.com/.

20 Tosin Daniel Oyetoyan, Bisera Milosheska, Mari Grini, and Daniela Soares Cruzes. Myths and Facts About Static Application Security Testing Tools: An Action Research at Telenor Digital. *International Conference on Agile Software Development*, pages 86–103, May 2018. Springer, Cham. doi:10.1007/978-3-319-91602-6_6.

21 Suri Patel. 2019 Global Developer Report: DevSecOps finds security roadblocks divide teams. Online, Accessed 18 July 2020. URL: https://about.gitlab.com/blog/2019/07/15/global-developer-report/.

22 Tim Rietz and Alexander Maedche. LadderBot: A Requirements Self-Elicitation System. *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 357–362, September 2019. IEEE, Jeju, South Korea. doi:10.1109/RE.2019.00045.

23 Gaigai Tang, Lianxiao Meng, Shuangyin Ren, Weipeng Cao, Qiang Wang, and Lin Yang. A Comparative Study of Neural Network Techniques for Automatic Software Vulnerability Detection. *CoRR*, abs/2104.14978, 2021. arXiv:2104.14978.