LLM-based Vulnerability Detection

Hongping Li, Li Shan
School of Computer Science and Engineering
University of Electronic Science and Technology of China
Chengdu, China

***Abstract*—The emerging Large language models (LLMs) are increasingly used as a revolutionised tools in many industry. In cyber security, the LLMs can offer innovative ways to detect, identify, and mitigate vulnerability through their natural language processing capability. This work aims to fine tune pre-trained LLMs to detect anomalies and vulnerability by analysing vast amounts of data. A ChatGPT 3.5 model based vulnerability detector were developed that can conduct common vulnerability analysis. Experimental results demonstrate the effectiveness of proposed solution.**

***Index Terms*—Cyber attack, Detection, LLM, GPT**

I. Introduction

The cyber security landscape for the Internet of Things (IoT) are rapidly evolving and becoming increasingly complicated and capable. With the rise of new technology, such as artificial intelligence (AI), 5G, deep learning, etc., new cyber threats targeting IoT devices increased significantly in 2023, making them a prime target for cyber attacks [1]. Top common cyber attacks include DDoS, Man-in-the-middle (MITM), phishing attacks, whale-phishing attack, spear-phishing attacks, etc., in which 82% involved the 'human factor' and 47% of all cyber attacks are involved with personally identifiable information [2], [3].

It is reported that over 80% of organisations have implemented the IoT, in which 20% of them have detected IoT-based attacks In IoT, threats are changing more complicated and sophisticated, e.g., *large-scale data breaches, ransomware attacks, and phishing attacks* are now common and devised with extraordinary sophistication [4]. Due to lack proper security measurements, making these IoT devices vulnerable to attacks. The attack surface has expanded significantly in IoT and it is more challenging to protect IoT systems [5]. The AI plays a key role in improving IoT security measures, including threats identification, cyber attack analysing, anomaly detection, etc. On the other hand, AI techniques have been used to create more sophisticated attacks, including social engineering attacks, large scale DDoS attacks, AI-empowered malware (e.g. Deeplocker) and more [6].

In the past decades, new approaches have been developed against cyber attacks in IoT environment, including: (1) Network segmentation, separating IoT devices from Internet to prevent attacks; (2) Device management, appropriate security settings and updated with latest firmware can enhance security of IoT devices; (3) Apps/users behavioral analysis, detecting abnormal activity in IoT can identify and prevent AI-based attacks; (4) Using machine learning algorithms to detect and analyse potential thrats by detecting anomalies IoT.

The generative artificial intelligence (GenAI) and large language models (LLMs) show great potential in advancing IoT cyber security. Specifically GenAI and LLMs can help well understand the potential threats in different ways:

- Vulnerability detection, LLMs can be used to scanning and filtering the IoT vulnerabilities, LLM models can scan for vulnerabilities in IoT system development, including programming language, system design, and application developments;
- LLMs can be used to analyse APIs of applications and create rules against reverse engineering frameworks;
- LLMs can be used in threats hunting queries for malware analysis tools (e.g., YARA, etc.), using LLMs to develop threat hunting queries can significantly enhance operation efficiency and expedite response times;
- The GenAI and LLMs can be used to detect AI-generated malware and text, which can make IoT system be able to spot phishing emails, and other flags;
- Assisting existing cyber security tools, *e.g., Microsoft 365 defender, Google' AI Red Team, etc.*, LLMs can help generate secure and flawless code.

The continuously evolving LLMs please a central role in enhancing IoT security: (1) assessing the impact of simulated attacks in IoT and identify potential ways to increase resilience against these attacks; (2) analyse the resilience of new AI detection and prevention capabilities of IoT system, and probe how an attacker might bypass them; (3) simulate IoT attacks and raise awareness among relevant stakeholders. The main contribute of this work can be summarised as:

(1) In this work, a LLM based cyber security vulnerability detection framework was proposed, which includes data collection, LLMs selection, fine-tuning of LLMs, and analysis result generation;

(2) Use cases for vulnerability detection in both web applications (Awesome Webext) and common buffer over flow were investigated and the results demonstrated that the proposed framework can help provides detailed vulenrability reasoning, decision making, solution, and potential threats.

II. Related Works

The emerging LLMs were used to enhance the cyber security by offering innovate ways to identify and mitigate vulnerabilities. LLMs algorithms can be employed in features recognition and detection of cyber security vulnerabilities through their natural language processing capabilities. Analyzing vast amounts of security reports, system event logs, network traffic, and even code repositories, pre-trained LLMs can automatically detect and recognise patterns and anomalies in text, ultimately enhancing the efficiency and accuracy of vulnerability scanning [7].

Altin *et al.* developed a LLM tool using ChatGPT-3.5 to analyse Common Vulnerabilities and Exposures (CVE) to explain and provide better understanding for CVE Proof of Concept (PoC). A simplified model, ChatGPT 3.5-turbo model was developed that can scan and understand flaws, e.g., the token limit of ChatGPT's long-term memory [8].

Mulgrew*et al.* developed a zero-day virus scanner using LLM prompts, which uses ChatGPT to refactor the Auyer's steganographic function library and create steganography function for local applications without the need to call external library [9], [10]. Luoto *et al.* developed a new way to use LLMs (e.g., ChatGPT, llama2) to reverse minified javascript[1], in which the LLMs were used as a complex markov chain to guess the next word [11], [12].

Chris *et al.* developed a GPT-3 based vulnerability detector using text-davinci-003, which can process hundred lines of code (including js, Flask, python, C library, etc.) per request and accurately detect security vulnerabilities [13]. Nagy *et al.* combined LLMs (e.g., GPT4) in a cloud based code security scanning tools (Semgrep) to identify secuirty flaws and notify developers. In this work, the pre-trained LLMs were used to create rules to identify and predict possible flaws to automatically fix and reduce the false positive alerts.

In cyber security industry, the rather than use LLMs to analyse security issues, a lot of research efforts have been conducted to identify vulnerabilities in LLMs, which is very important to guarantee the security of these models and testing frameworks. The OWASP listed 10 for LLM include ten most critical vulnerabilities that may affect LLMs, including prompt injection, insecure output, training data poisoning, model DoS, supply chain vulnerabilities, etc. [2].

III. LLM based Vulnerability Detection

The trained LLMs are able to understand and generate natural language, which can learn the patterns and relationships from massive datasets of code and text.

A. LLM based Cyber Vulnerability Detection

Usually training a LLM involves following stage: *pre-training, supervised fine-tuning, reward modeling, and reinforcement learning*. This work focuses on supervised fine-tuning state, in which additional question-answer pairs are introduced to refine the pre-trained LMMs. Supervised fine-tuning a model for a specific task, *e.g.*, text summarising or translating natural language to database queries. This work will use base LLMs (*e.g., GPT 3.5, GPT-4, etc.*) and the main procedures include:

1) LLMs selection. Based on the objectives, tasks, costs, of vulnerability, it is crucial to choose an appropriate LLM before finetuning for a certain cyber vulnerability detection scenario. In this work, we use the two use base LLMs GPT 3.5, GPT-4;
2) Supervised fine-tuning, using techniques LoRa (Low-Rank Adaptation) or QLoRA (Query-LoRA). Usually it needs an LLM to construct a training dataset to fine-tune desired LLM [14];
3) Feed LLM with prompts engineering, including construct an evaluation dataset, build prompts based on LLMs, and deploy the prompts to production and track user input and feedback;
4) Vector similarity search at query time to produce relevant chunks of text;
5) Multi-hop question-answering can be used to generate an accurate answer, in which a question can be broken down into multiple sub-questions.

[1]https://thejunkland.com/blog/using-llms-to-reverse-javascript-minification

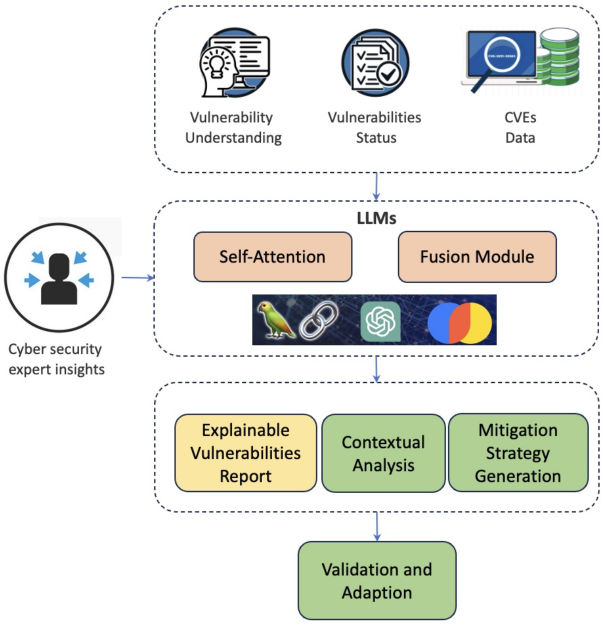

Vulnerability Understanding
Vulnerabilities Status
CVEs Data
LLMs
Self-Attention
Fusion Module
Cyber security expert insights
Explainable Vulnerabilities Report
Contextual Analysis
Mitigation Strategy Generation
Validation and Adaption

Fig. 1. LLM based Vulnerability Detection using LLM

In this work we propose newly enabled attack vectors and techniques and provide demonstrations of each in this repository [15].

B. Vulnerability Scanning and Filtering using LLMs

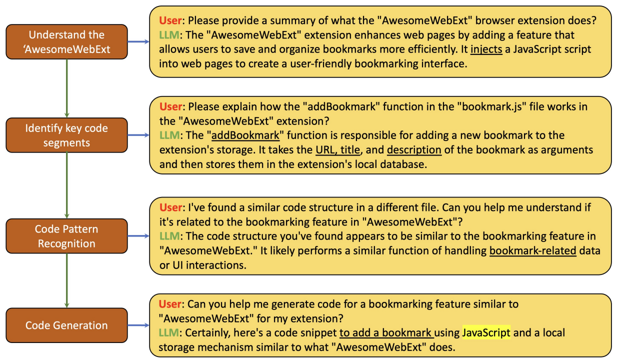
The LLMs are able to detect potential vulnerabilities in the model prompt, which has been widely used in code analysis and security risk evaluation. The LLMs shows great potential in vulnerability scanning of *Web application, IoT systems, etc.*, as an important complementary to existing vulnerability scanning and detection tools. Specifically, based on the request and response pairs generated based on the collected data, LLMs can be used to detect logical vulnerabilities using following steps:

1) **Pattern recognition**, LLMs are trained on a wide range of text, including code samples and discussions about

software security. The LLMs are able to recognise patterns in code that are known to be security risks, e.g., buffer overflow protection, absence of input validation, etc.

2) **Vulnerability analysis**. LLMs are trained on a wealth of text, including vulnerabilities in IoT, so it can identify common vulnerabilities;
3) **Language understanding**. LLMs are able to understand the context and meaning of code, e.g., it can understand the 'scanf' is used for input that associated with unbounded string input;
4) **Inference**. LLMs can make inferences about the potential consequences of code.
5) **General knowledge**. LLMs have access to general knowledge up to their knowledge cutoff date.

Assisting key vulnerability scanning tools, such as *Nmap, OpenVAS, Nessus etc.*, fine tuned LLMs can generate rules for understanding, detection, and prompts.



User: Please provide a summary of what the "AwesomeWebExt" browser extension does?
LLM: The "AwesomeWebExt" extension enhances web pages by adding a feature that allows users to save and organize bookmarks more efficiently. It injects a JavaScript script into web pages to create a user-friendly bookmarking interface.

User: Please explain how the "addBookmark" function in the "bookmark.js" file works in the "AwesomeWebExt" extension?
LLM: The "addBookmark" function is responsible for adding a new bookmark to the extension's storage. It takes the URL, title, and description of the bookmark as arguments and then stores them in the extension's local database.

User: I've found a similar code structure in a different file. Can you help me understand if it's related to the bookmarking feature in "AwesomeWebExt"?
LLM: The code structure you've found appears to be similar to the bookmarking feature in "AwesomeWebExt." It likely performs a similar function of handling bookmark-related data or UI interactions.

User: Can you help me generate code for a bookmarking feature similar to "AwesomeWebExt" for my extension?
LLM: Certainly, here's a code snippet to add a bookmark using JavaScript and a local storage mechanism similar to what "AwesomeWebExt" does.

Fig. 2. ”AwesomeWebExt” Add-on reverse engineering

C. Reversing add-ons, analysing APIs using LLM

LLMs can be used to develop rules and reverse popular add-ons based on reverse engineering frameworks like IDA and Ghidra [16]. In security software development, LLMs (*e.g., GPT-3, GPT-4, etc.*) can help automatically build rules for identifying specific patterns or behaviors in code. LLMs are able to translate natural language descriptions into actionable code rules to achieve use rules. Analysing *documentation, readme files, source code, comments, etc.*, LLMs can understand their functionality, usage, and dependencies, and further highlight how they work and how to potentially reverse-engineer similar functionality. Fig. 2 shows an example to do reverse Add-on ”AwesomeWebExt”, it includes following procedures:

1) Understanding the ”AwesomeWebExt”, LLM provides a summary of AwesomeWebExt, including the purpose, key features, and how it interacts with web pages;
2) Identifying the key code segments, using the pre-trained LLM key function ”addBookmark” can be identifyed in 'bookmark.js', highlighting the functions, classes, code segments;
3) Recognising code pattern, the LLMs can identify similar code patterns 'structure' in bookmark-related data or UI interactions;
4) Code Generation, the LLMs are able to create code snippets for the required functionality.
5) Debugging and troubleshooting.

Like in the context of code analysis, the LLMs can play a crucial role in vulnerability detection phishing detection and malicious content detection. The innate natural language understanding capabilities of LLMs also prove invaluable in facilitating regulatory compliance and ethical considerations, particularly in the realm of privacy-enhancing techniques.

IV. Validation

To validate the above proposed framework, two different LLMs were used to validate the buffer overflow vulnerability. The code listed in Table. I were prompted into LLMs, the LLMs first extract relevant code snippets (e.g., the function name $scanf$, variables, etc.) Then irrelevant information will be removed and the code will be reformatted ensure the code is well structured that can be feed into LLMs.

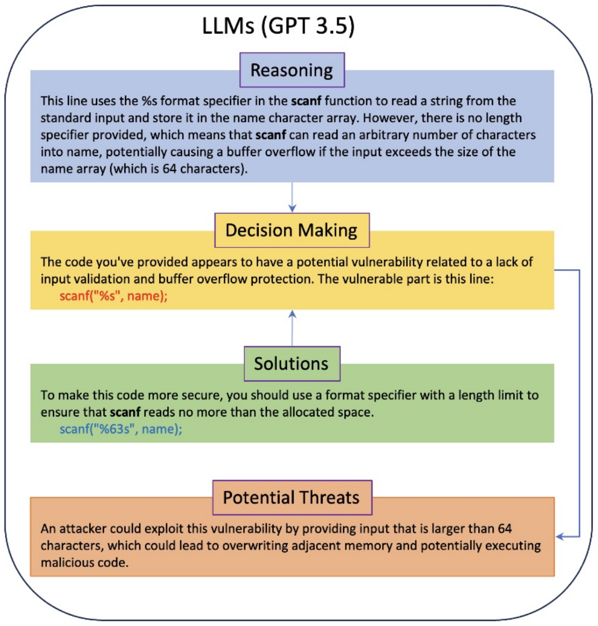
```
int _tmain(int argc, _TCHAR* argv[])
{
        char name[64];
        printf("Enter your name: ");
        scanf("%s", name);
        Sanitize(name);
        printf("Welcome, %s!", name);
        return 0;
}
```

In this example, ChatGPT 3.5 and ChatGPT 4.0 were used to analyse the extracted text, both will assist in identifying c language programming patterns. Specifically, the input parameters, variables, key functions ”$printf()$”, ”$scanf()$”, ”$Sanitize()$”, will be analysed and provide insights into the their functionality.

Fig. 3 (a) shows the analysis procedures using GPT 3.5, including detailed reasoning for the code, decision making, solutions, and finally potential threats were generated that can help user to update security strategy to mitigate the vulnerability. Different with GPT 3.5, the GPT 4.0 in Fig. 3(b) made reasoning based on the 63 characters and provided more useful solutions for fixing this concerns, e.g., it suggests to use '$fgets$' or '$getline$' instead of '$scanf$' in the context of code.

V. Conclusion

While language models like ChatGPT can provide valuable insights and assist with code analysis, it's important to note that they are not a replacement for specialized code analysis tools and manual review by experienced developers and security experts. They may not have a deep understanding of specific programming languages, code execution, or the intricacies of code vulnerabilities. Therefore, their analysis



This line uses the %s format specifier in the **scanf** function to read a string from the standard input and store it in the name character array. However, there is no length specifier provided, which means that **scanf** can read an arbitrary number of characters into name, potentially causing a buffer overflow if the input exceeds the size of the name array (which is 64 characters).

The code you've provided appears to have a potential vulnerability related to a lack of input validation and buffer overflow protection. The vulnerable part is this line:

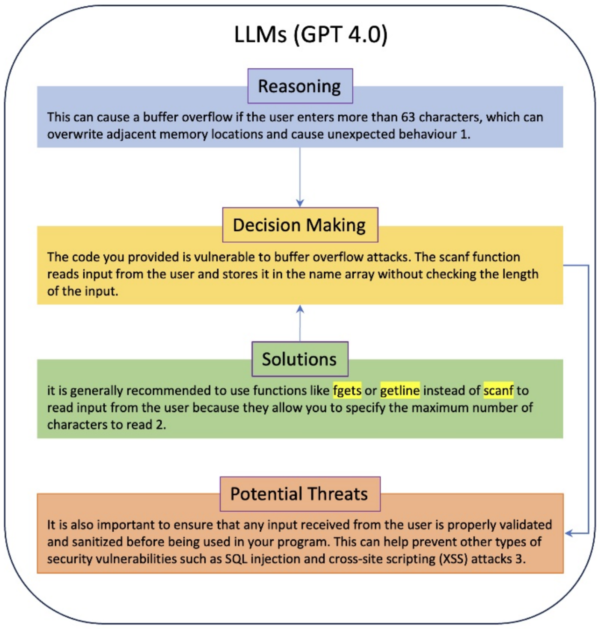
```
scanf("%s", name);
```

To make this code more secure, you should use a format specifier with a length limit to ensure that **scanf** reads no more than the allocated space.

```
scanf("%63s", name);
```

An attacker could exploit this vulnerability by providing input that is larger than 64 characters, which could lead to overwriting adjacent memory and potentially executing malicious code.

(a) Using GPT 3.5



This can cause a buffer overflow if the user enters more than 63 characters, which can overwrite adjacent memory locations and cause unexpected behaviour 1.

The code you provided is vulnerable to buffer overflow attacks. The scanf function reads input from the user and stores it in the name array without checking the length of the input.

it is generally recommended to use functions like fgets or getline instead of scanf to read input from the user because they allow you to specify the maximum number of characters to read 2.

It is also important to ensure that any input received from the user is properly validated and sanitized before being used in your program. This can help prevent other types of security vulnerabilities such as SQL injection and cross-site scripting (XSS) attacks 3.

(b) Using GPT 4.0

Fig. 3. Buffer Over Flow Vulnerability Detection using LLM

should be used as a supplementary tool in the code analysis process.

References

[1] F. Bell, R. Pilbery, R. Connell, D. Fletcher, T. Leatherland, L. Cottrell, and P. Webster, "The acceptability and safety of video triage for ambulance service patients and clinicians during the covid-19 pandemic," *British paramedic journal*, vol. 6, no. 2, pp. 49–58, 2021.

[2] Y. Zhang, W. Song, Z. Ji, N. Meng *et al.*, "How well does llm generate security tests?" *arXiv preprint arXiv:2310.00710*, 2023.

[3] H. Chang, F. Yu, J. Wang, D. Ashley, and A. Finkelstein, "Automatic triage for a photo series," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, pp. 1–10, 2016.

[4] M. Pirrung, N. Hilliard, A. Yankov, N. O'Brien, P. Weidert, C. D. Corley, and N. O. Hodas, "Sharkzor: Interactive deep learning for image triage, sort and summary," *arXiv preprint arXiv:1802.05316*, 2018.

[5] T. Kaluarachchi, A. Reis, and S. Nanayakkara, "A review of recent deep learning approaches in human-centered machine learning," *Sensors*, vol. 21, no. 7, p. 2514, 2021.

[6] M. Mozes, X. He, B. Kleinberg, and L. D. Griffin, "Use of llms for illicit purposes: Threats, prevention measures, and vulnerabilities," *arXiv preprint arXiv:2308.12833*, 2023.

[7] Y. Wu, N. Jiang, H. V. Pham, T. Lutellier, J. Davis, L. Tan, P. Babkin, and S. Shah, "How effective are neural networks for fixing security vulnerabilities," *arXiv preprint arXiv:2305.18607*, 2023.

[8] Tin-Z. (2023) Escaping chatgpt memory. [Online]. [Online]. Available: https://tin-z.github.io/chatgpt/go/cve/2023/04/14/escaping_chatgpt_memory.html

[9] A. Qammar, H. Wang, J. Ding, A. Naouri, M. Daneshmand, and H. Ning, "Chatbots to chatgpt in a cybersecurity space: Evolution, vulnerabilities, attacks, challenges, and future recommendations," *arXiv preprint arXiv:2306.09255*, 2023.

[10] Forcepoint X-Labs. (2023) Zero day exfiltration using chatgpt prompts. [Online]. [Online]. Available: https://www.forcepoint.com/blog/x-labs/zero-day-exfiltration-using-chatgpt-prompts

[11] CiteDrive. (2023) How to cite a website in latex using bibtex and biblatex. [Online]. [Online]. Available: https://citedrive.medium.com/how-to-cite-a-website-in-latex-using-bibtex-and-biblatex-763770172cb5

[12] The Junkland. (2023) Using llms to reverse javascript minification. [Online]. [Online]. Available: https://thejunkland.com/blog/using-llms-to-reverse-javascript-minification

[13] Github repository: chris-koch-penn. [Online]. [Online]. Available: https://github.com/chris-koch-penn/

[14] C. for Research on Foundation Models (CRFM). (2023) Alpaca: Alleviating language priors with accuracy certainty analysis. [Online]. [Online]. Available: https://crfm.stanford.edu/2023/03/13/alpaca.html

[15] B. Greshake. (2022) Large Language Models: Looking Back and Ahead. [Online]. [Online]. Available: https://github.com/greshake/llm-security

[16] Y. Huang, S. Gupta, M. Xia, K. Li, and D. Chen, "Catastrophic jailbreak of open-source llms via exploiting generation," *arXiv preprint arXiv:2310.06987*, 2023.