electronics

Article

Using Generative AI Models to Support Cybersecurity Analysts

Štefan Balogh, Marek Mlynček, Oliver Vraňák and Pavol Zajac *

Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava, Ilkovicova 3, 841 04 Bratislava, Slovakia; stefan.balogh@stuba.sk (Š.B.)
* Correspondence: pavol.zajac@stuba.sk

Abstract: One of the tasks of security analysts is to detect security vulnerabilities and ongoing attacks. There is already a large number of software tools that can help to collect security-relevant data, such as event logs, security settings, application manifests, and even the (decompiled) source code of potentially malicious applications. The analyst must study these data, evaluate them, and properly identify and classify suspicious activities and applications. Fast advances in the area of Artificial Intelligence have produced large language models that can perform a variety of tasks, including generating text summaries and reports. In this article, we study the potential black-box use of LLM chatbots as a support tool for security analysts. We provide two case studies: the first is concerned with the identification of vulnerabilities in Android applications, and the second one is concerned with the analysis of security logs. We show how LLM chatbots can help security analysts in their work, but point out specific limitations and security concerns related to this approach.

Keywords: LLM; cybersecurity; vulnerability assessment

check for updates

Citation: Balogh, Š.; Mlynček, M.; Vraňák, O.; Zajac, P. Using Generative AI Models to Support Cybersecurity Analysts. *Electronics* **2024**, *13*, 4718. https://doi.org/10.3390/electronics13234718

Academic Editors: Aryya Gangopadhyay, Shanshan Zhao and Shancang Li

Received: 30 September 2024
Revised: 29 October 2024
Accepted: 27 November 2024
Published: 28 November 2024

1. Introduction

Artificial Intelligence (AI) and the usage of generative large language models (LLM for simplicity) are currently penetrating many areas of human activities, including areas that work with sensitive and private information. From the scientific point of view, it is thus important to research the LLM applications and their interaction with current security technologies.

In the area of cybersecurity, we can identify many potential domains where LLMs can help to increase security and protect user privacy (defensive LLM applications). On the other hand, there is also a trend of using LLMs for offensive purposes. According to [1], there are several advanced offensive LLM models for cybercriminals. The capabilities of these tools include generating a reverse shell, building custom malware, or even locating people based on an image. An important research direction in defensive cybersecurity should be the development of AI/LLM-based tools and methods that can provide experts with tools for preventing offensive AI/LLM tools. In this article, however, we restrict our research focus mainly to the topic of using LLMs to support defensive security specialists, namely in providing tools for automatic security flaw detection and simplifying the analysis of security-related data.

Note that in many areas of defensive security, LLMs have already been tested, and are shown to be efficient and helpful [1–4]. New capabilities of AI are supposed to improve malware and incident analysis, even if the organization does not have a team of experts. In our opinion, this will however incite malware creators to improve their obfuscation techniques, and specifically target known AI-based detectors' weaknesses (which requires the development of new detection techniques, such as [5]). Another application of AI/LLM tools is the detection of (code) vulnerabilities and the automation of penetration testing (see e.g., PentestGPT [6]). According to expert opinion, security analysts would welcome AI assistance in generating playbooks for incident reactions, or attack scenarios/playbooks, but we have not found current research papers in this area (to date).

This research paper focuses on two case studies of LLM applications that were developed during our master theses at our university. We have tried to use LLMs to support two types of security tasks:

1. The penetration testing of Android applications [7];
2. The detection and classification of incidents from security logs [8].

Our aim is in each case to present core parts of the solution and the potential of LLMs, demonstrated in small practical tests of the developed tools. We also want to show the potential risks and problems involved when using LLMs in these specific security tasks.

2. Related Works

Large language models (LLMs) have shown significant potential in the cybersecurity domain. They are used for various tasks that include Log Analysis, Vulnerability Detection, Cyber Threat Hunting, Code Security, and Offensive Security. LLMs like BERT, RoBERTa, and GPT-2 have been fine-tuned to analyze application and system log files, significantly improving the detection of security anomalies [9]. Li and Shan [10] have investigated the capabilities of ChatGPT 3.5 and GPT 4.0 to detect the buffer overflow vulnerability. Other LLMs are being utilized to enhance cyber-threat hunting by processing vast amounts of data to uncover hidden patterns and detect anomalies. This application helps organizations respond to emerging threats in real time, although challenges related to bias, privacy, and computational efficiency remain [11].

LLMs can also be used for offensive purposes, such as generating malicious content and discovering vulnerabilities. For instance, Net-GPT, an LLM-empowered chatbot, has been developed to perform Man-in-the-Middle (MITM) attacks, demonstrating the dual-use nature of these models [12].

LLMs are often used as coding assistants. Studies indicate that while these tools can introduce some security vulnerabilities, the overall impact on code security is relatively small, with AI-assisted users producing critical bugs at a rate only slightly higher than non-assisted users [13].

Many efforts are being made to develop domain-specific LLMs for cybersecurity, such as HackMentor, which has shown significant performance improvements in handling cybersecurity prompts compared to general LLMs [14].

Mohammed et.al. [15] explore the impact of ChatGPT on cybersecurity including constructing honeypots, code security enhancement, misuse in malware development, vulnerability exploration, contribution to cybersecurity policies, vulnerability exploration, propagating disinformation, attacks on industrial systems, changing the cyber threat landscape, adapting cybersecurity strategies, the utilization of AI for defense, and human-centric training evolution.

The latest release of CyberMetric includes a novel benchmark dataset that evaluates the level of expertise of LLMs in cybersecurity [16].

ChatGPT can be very helpful in simplifying the process of launching complex phishing attacks, even for less skilled hackers, by helping them to manage the configuration setup, bypass the filters used by ChatGPT, and deploy a fully automated phishing kit [17]. It highlights how difficult it is to guard against the malicious usage of GenAI capabilities and the necessity to enhance countermeasures within AI systems.

Scanlon et.al. [18] provided a thorough analysis of the literature on LLMs from a three-fold perspective: beneficial security applications (e.g., vulnerability detection, secure code generation), adverse implications (e.g., phishing attacks, social engineering), and vulnerabilities (e.g., jailbreaking attacks, prompt attacks), along with their corresponding defensive measures. They illuminated the complexity of LLM applications, pointing out both the important constraints and potential prospects that come with GenAI (as it is now).

3. Tools and Methods

In this section, we summarize the tools and methods used for our research. This includes not only large language models but also security tools. Note that in our research we used both commercial and open LLMs, but always in just a black-box way, with no custom training. Similarly, the security tools were used for specific tasks and were unmodified. We have however added custom tools to modify and filter the outputs of the security tools to present the data to the LLMs. This is mostly due to the large amount of data outputs produced by the analytical tools. It would be possible to use LLMs to analyze raw data, but this would incur significant additional costs (in terms of both time and USD cost per token).

Our research method consisted of creating and testing two different solutions for common security tasks, and testing whether current LLMs can provide correct results and support both security tasks.

The first solution is focused on vulnerability detection in mobile applications. The system should be able to identify a wide spectrum of security vulnerabilities, from basic problems to advanced ones. The ultimate goal is to automate the process to minimize the impact of human factors that limit the speed and scope of penetration tests. We hypothesize that integrating LLMs (and other similar AI tools) can help to use the logic of examined applications to identify security context, and potentially to find new vulnerabilities which the standard tools have not been able to detect.

The second solution aims to use logs from detection and monitoring tools (Suricata, Sysmon) to verify whether a reported alert is indeed a security incident. We hypothesize that LLMs can help to identify and categorize security incidents based on the provided logs (and context data). An interesting question is whether LLM can correlate data from different sources and use this information to filter out false alerts. Furthermore, LLM should be able to identify attacks and use specified security frameworks (in our case Mitre) to provide an analysis of the attack and classification of incidents. Finally, to emulate the whole process of security log analysis, the model should be able to find all potential indicators of compromise, which can be used as proof of malicious activities.

3.1. Large Language Models

When selecting LLMs for this research, we have used benchmarks from [19], and started the study with models `GPT-4` or `GPT-3.5-turbo`. Both these models are commercial and have been acquired from the same vendor (OpenAI); thus, we have included models from other companies for comparison. Thus, in the full study, we have included the following models:

- GPT-3.5-turbo-0125—older commercial model from Open AI;
- GPT-4—newer commercial OpenAI model;
- Mistral-7b-instruct—model developed by Mistral AI;
- Claude-3-haiku—model from Anthropic;
- Claude-3-opus—newer version of Claude-3 model;
- Gemma-7b-it—model from Google.

3.2. Security Frameworks

Our research goal was to support security analysts in the work by correctly identifying security problems from the analytical tools with the help of LLM. The results of the LLM should provide security context according to a selected respected security framework. We have used two different security frameworks in our research: OWASP Mobile Top 10 and Mitre ATT&CK.

3.2.1. OWASP Mobile Top 10 2024

For our mobile application vulnerability research, we have used OWASP Mobile Top 10 2024 framework https://owasp.org/www-project-mobile-top-10/ (accessed on 31 March 2024). This framework is an important source of current security information and recommendations for both developers and security experts, focusing on security risks relevant to mobile applications. The framework is a result of an analysis of security threats and points out the 10 most frequent categories of vulnerabilities that should be taken into account when developing, implementing, and testing mobile applications. According to OWASP, mobile application testing includes a series of manual and automated tasks focused on the identification and removal of security problems such as Insecure Data Storage, Insufficient Cryptography, and others. Our research aims to help the security analyst to identify potential security problems with the help of LLM. The LLM should also be able to correctly identify the OWASP risk category so that appropriate recommendations can be found.

3.2.2. Mitre ATT&CK Framework

Mitre ATT&CK categorizes tactics (goals) and techniques (specific methods to reach these goals) of the attacker on various platforms and operating systems. As of March 2024, the Mitre https://attack.mitre.org/ (accessed on 31 March 2024) database contained 14 basic tactics and several techniques for each of the tactics.

3.3. Security Analytical Tools

There is a large number of analytical tools that help security experts in their work. Most of the tools provide large amounts of data. Typically, a security analyst needs to examine these data and make their security-relevant conclusions. We aim to support this work with LLMs in general, but for practical research purposes, we need to restrict our research to specific tools. For mobile application security research, we have used the Mobile Security Framework, along with the Semgrep tool. For our analysis of security logs, we have used logs from Suricata and Sysmon tools.

3.3.1. Mobile Security Framework—MobSF

Mobile Security Framework, https://github.com/MobSF/Mobile-Security-Framework-MobSF (accessed on 31 March 2024), is an open-source complex tool for automated security analysis of mobile applications. This tool can do both static and dynamic analysis of applications for Android, iOS, and Windows. MobSF aims to integrate various test techniques into a single centralized platform.

MobSF allows us to quickly scan APK files of Android applications to identify vulnerabilities including both vulnerable code and dangerous permissions. In our solution, we use MobSF reports to provide contextual and specific information as an input of our analysis. Namely, we use entry point and application segment identification, as well as logs from application runtime. For the decompilation of Android applications, we use the `jadx` subprocess of the MobSF.

3.3.2. Semgrep

Semgrep, https://semgrep.dev/ (accessed on 31 March 2024), is a tool for static code analysis. It helps in the effective search and identification of programming mistakes and security vulnerabilities in the source code of the application. We can create complex rules for these tasks that reflect our security requirements. We use this tool to localize suspicious places in the source code of the application and to provide context for the language model that analyzes both the results from MobSF and Semgrep (in a specified format and query language).

3.3.3. Suricata

Suricata is an IDS/IPS (intrusion detection/prevention system), which can efficiently analyze network traffic, and identify (and potentially prevent) security incidents in real-time. For each detection of a potential network anomaly, Suricata generates a flow. This is a collection of network packets which meet certain criteria. Each flow contains packet metadata, such as source and destination IP addresses, ports, and protocols. Suricata can issue warnings if the flow matches some predefined rules. In our solution, we focus on the generated warnings, which also contain flow information, as well as warning significance.

3.3.4. Sysmon Monitor

Sysmon, known also as System Monitor, is a system service in Windows OS. Sysmon has a key role in monitoring and logging all activities in the system. The Sysmon reports include detailed information about various system operations including process creation, network connections, and filesystem changes. We include selected (processed) Sysmon logs in our solution as an input to LLM in support data obtained by Suricata.

4. Identification of Vulnerabilities in Android Applications

The first case study concerns a solution that uses generative large language models for the automated penetration testing of Android applications. The system aims to identify a wide spectrum of application vulnerabilities (in practice, we focus on OWASP Mobile Top 10 vulnerabilities). The system was developed as part of the master thesis [7]. An application demo is available with the source code at https://github.com/StefanB172/cybersecurity-LLM (accessed on 26 November 2024).

The system is composed of multiple interconnected components, schematically shown in Figure 1. The inputs of the system have two parts: the APK file (application itself) and a log file produced by the application during its use.

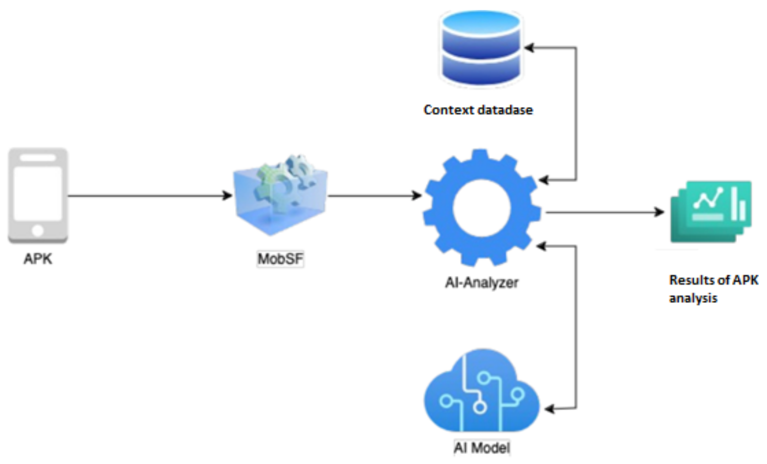

Figure 1. Use of MobSF in AI-assisted analysis.

The main idea of the analytic process can be simplified to a sequence of the following steps:

1. The APK file is decompiled from DEX to obtain the Java source code.
2. The Android manifest (in XML) format is extracted, providing metadata such as component definitions, used schemas, permissions, and others.
3. Application logs are extracted from the system and merged with source code and application metadata.
4. The data are presented to LLM with a specific query format, asking about potential vulnerabilities.
5. The LLM response is sent to the "Output parser" that interprets the results and presents them to the user (which can be either a human analyst or additional security subsystems).

In practice, including the whole source code, metadata, and logs in the LLM query is not practical, especially if we consider that model pricing is based on the number of processed tokens. The application compresses APK data with the help of a Mobile Security Framework (MobSF). The MobSF performs a set of statically defined tests (using RegEx rules). The data obtained from the MobSF are then enhanced by APK metadata from the manifest and from application logs. Application logs are obtained using standard tool `logcat`, which are then sanitized by the system (namely duplicates and near duplicates are removed, leaving unique log entries).

The pre-processed data are processed by the main AI-analyzer component. The schematic work of this component is summarized in Figure 2. Using LangChain library, https://www.langchain.com/ (accessed on 31 March 2024), we insert the input data into a specific LLM chatbot query template. The template contains a preamble to provide the chatbot with context (penetration testing), then inserts a Semgrep rule and its results, along with additional context from application metadata and logs. The model is then asked whether the provided data introduce a security risk. This question is repeated with every Semgrep rule, instead of providing all Semgrep results as one query.

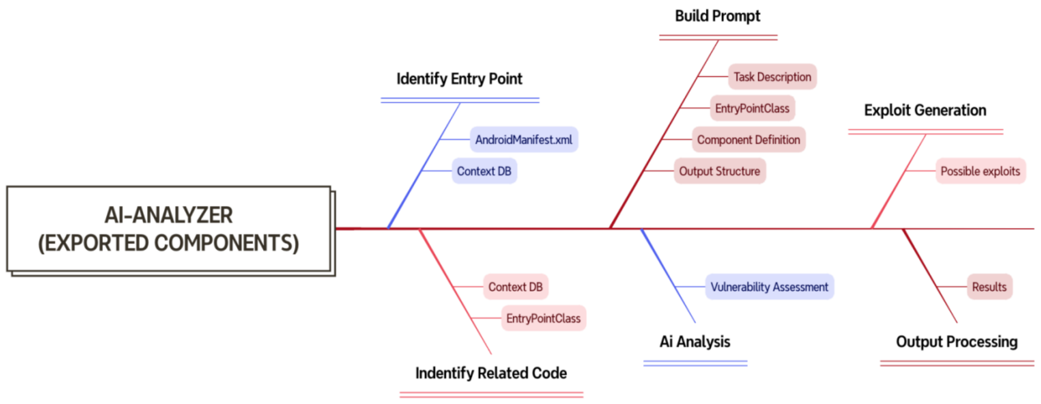

Figure 2. AI-analyzer component.

Note that this final architecture is significantly different from just asking the model about application security from all data. Our process focuses on individual potential vulnerabilities as identified in the OWASP framework, supported by specific Semgrep rules and extra data from the manifest and logs. The role of the AI is to interpret the results, instead of providing them itself from raw data. Thus, the process is more robust and better reflects a systematic penetration test workflow.

4.1. *Test Results and Discussion*

We have prepared Semgrep rules for the following list of vulnerability types:

- Insecure Logging;
- Insecure Shared Preferences;
- Hardcoded Credentials;
- Root Detection;
- Arbitrary Code Execution;
- Secure Flag Bypass;
- Certificate Pinning Bypass;
- Insecure Broadcast Receiver;
- Insecure Service;
- Insecure Content Provider;
- Insecure Activity;
- Deep Link Exploitation;
- SQL Injection;
- Vulnerable WebView;
- Smali Patching;
- Native Libraries.

We have tested our system on a list of open-source Android applications with known vulnerabilities (the number in brackets is the number of known vulnerabilities from our list):

- AllSafe (11);
- Insecureshop (13);
- AndroGoat (13);

- MASTG Playground (9);
- InjuredAndroid (5);
- InsecureBankV2 (8);
- Damn Vulnerable Bank (7).

Unlike other examples, the ”Damn Vulnerable Bank” application code is obfuscated, and as we will see, this obfuscation makes detecting vulnerabilities more difficult.

Note that application vulnerability testing is resource intensive, both on the pre-processing side and when using the chosen LLM (which also incurs additional costs per token). Thus our tests have only a small scope and a larger study would be needed to provide a more precise look into the efficacy of the method.

In the first test, we tried to analyze all the applications with the help of `GPT-3.5-turbo`. Each application was tested three times (model response is stochastic, and can be different for the same query). The results of the tests are summarized in Table 1.

Table 1. Analysis results, using `GPT-3.5-turbo` 3 times for each application.

Application	False-Positives	True-Positives	Success Rate
AllSafe	19	25/33	75%
Insecureshop	4	15/39	38%
AndroGoat	4	23/39	59%
MASTG Playground	2	20/27	74%
InjuredAndroid	16	6/15	40%
InsecureBankV2	24	18/24	75%
Damn Vulnerable Bank	6	10/21	48%

The success rate of the `GPT-3.5-turbo` was between 40% and 75%, but the numbers are not directly comparable as each application contained a different set of vulnerabilities (and different functionality). In some instances, the models have provided a large number of false positives (model detected vulnerability that was not present). The results indicate that LLMs can help in analytic work, but both the success rate is too low and the false-positive rate is too high for fully automatic deployment.

We have also tested different models (only once per application due to higher costs). Table 2 summarizes the data across all the seven tested applications.

Table 2. Analysis results, per model, cumulative data across all 7 applications/66 present vulnerabilities.

Model	False-Positives	True-Positives	Success Rate
GPT-3.5-turbo-0125	23	39/66	60%
GPT-4-turbo	15	48/66	73%
Claude-3-haiku-20240307	22	39/66	60%
Claude-3-opus-20240229	16	45/66	68%

We can see that newer models (`GPT-4-turbo` and `Claude-3-opus`) have a slighly higher success rate than the older models. The number of false positives was also decreased. Still, the automatic adaptation of this LLM-based method seems limited, even when the models are trained further, but newer models provide better support for the human analyst.

4.2. Cost-Benefit Analysis

When evaluating results, we should also take into account other practical aspects, such as the time required for the analysis and the incurred costs for LLM request processing. More complex applications with a larger source code and a higher number of functions require more time for detailed analysis and can activate more Semgrep rules (and subsequent LLM queries). An analysis of a single application can take hours of real-time in practice. On the other hand, the analysis can be run in the background without the involvement of the human operator, who can thus focus on the verification of the results and expert work.

During the testing phase, we have also tried to estimate the direct financial costs of this approach. The summary of the required time and financial costs incurred are presented in Table 3. Note that the average time for performing the analysis using Claude 3 models is artificially inflated. The cause of this is that the API calls for these models were limited by the number of tokens per minute (see https://docs.anthropic.com/claude/reference/rate-limits (accessed on 26 November 2024) for current information).

Table 3. Comparison of running times and price of performed experiments.

Model	Success Rate	Avg. Scan Time [min]	Avg. Scan Price [USD]
GPT-3.5-turbo-0125	60%	1.40	0.06
GPT-4-turbo	73%	2.37	0.83
Claude-3-haiku-20240307	60%	4.89 *	0.03
Claude-3-opus-20240229	68%	6.81 *	0.95

* scan time inflated due to token rate limiting.

Model GPT-3.5-turbo-0125 was the fastest one (with an average scan time of 1.4 min) and was relatively cheap (average price of USD 0. 06 per scan). This model is thus an "economy" option, but its success rate was lower than other models (only 60%). The best results were obtained by using the more costly (both in time and financial expression) model GPT-4-turbo. Note that model pricing and performance rapidly change, and these concrete figures should serve only as an illustration.

5. Analysis of Security Logs

The second case study concerns a solution that uses generative large language models for the analysis of security logs related to network monitoring-based intrusion detection. The system aims to help identify and categorize potential security incidents based on data generated by Suricata and Sysmon tools. The system was developed as part of the master thesis [8]. An application demo is available with the source code at https://github.com/StefanB172/cybersecurity-LLM (accessed on 26 November 2024).

The workflow of the system is similar to the previous solution and is shown in Figure 3. The main steps of the process are as follows:

1. Record processing—The application processing is based on Suricata alerts. The application collects all related network flows from Suricata as well as relevant logs from Sysmon. Relevancy is defined by some specific rules (such as all information about specific processes, users, or system events in a relevant time frame). All data are transformed into a single unified JSON file.
2. Correlation and reduction—The processed records are too large in practice to be presented to the LLM. At this stage, records are filtered and reorganized into more compact representations. Duplicities are filtered and some records are aggregated to provide derived aggregate information. The main filter is based on a specific time window; only related records within some smaller time frames around the incident are aggregated and sent to LLM. A smaller time window means a smaller size of query in tokens, but we also lose some potential context (especially if the attack takes longer time than the specified window size).
3. Building context—In this phase, we use LLM to compress the information from processed logs into a description of a potential incident as well as a set of Indicators of Compromise (IoCs). Namely, we ask a model to perform the following:

```
Your task is to:
- Analyze the Suricata and Sysmon logs if they have anything in common.
In case Sysmon/Suricata logs are missing, continue without them.
- Identify if the logs represent a security incident. Take into account
also the number of occurrences of each log.
```

```
- Describe the potential incident.
- Perform threat intelligence search on found IoCs.
Use only the provided information. Remember to reply shortly and
precisely.
```

This query is updated with the log data and sent to LLM. The response from LLM is then again parsed and stored in custom JSON format.

4. Classification—In the final stage, we use the LLM a second time. The input query now asks not only to identify potential incidents but also to classify the incidents within the Mitre ATT&CK framework. The query is accompanied by data from previous steps, including the response from the incident identification phase.

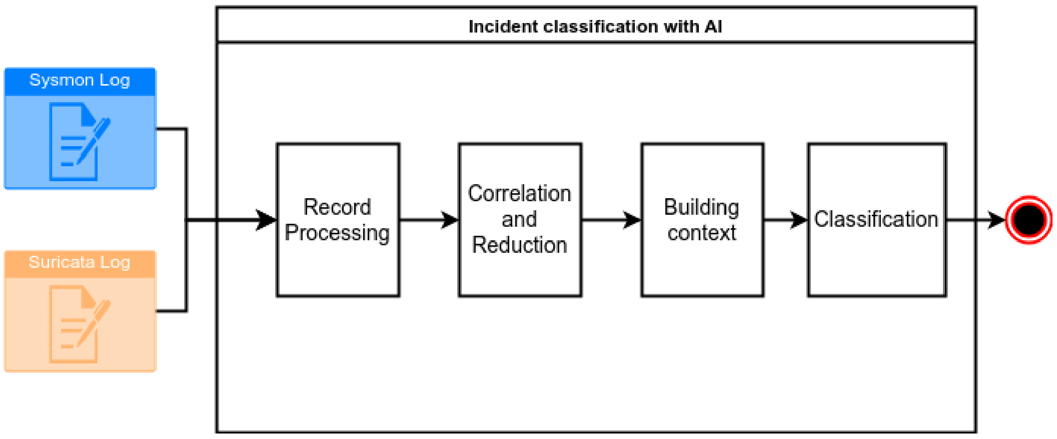

Figure 3. The architecture of security logs analyzer.

Note that this approach uses the LLM query in two stages. The first stage is used to identify potential incidents, and the second stage tries to classify them and provide more context about the attack. The second stage is only executed if some potential incidents are identified, and only for these incidents. This saves us some processing on Suricata alerts that are unlikely to represent real attacks.

Unlike the previous solution (mobile application vulnerabilities), the LLM processes all data related to the Suricata alert instead of trying to separately querying different attack types. It would be an interesting research question to compare this approach (category-based yes/no queries with specifically preprocessed data) on network monitoring problems, but the number of logs and attack categories is too large for us to process.

5.1. Test Results

We have tested the system with different LLMs on open data representing simulated cyberattacks. We were interested in whether LLMs can correctly identify the presence of a simulated attack, and then whether LLMs can correctly classify the attack into a correct Mitre ATT&CK tactic category.

During the testing phase, we simulated the following attacks:

1. Network sniffing (Mitm)—Mitre database: network sniffing (T1040), included in the Credential Access and Discovery tactics.
2. Denial of Service (DoS)—Mitre database: Network Denial of Service (T1498), included in Impact tactic.
3. Brute Force RDP (RDP)—The Mitre database does not contain the exact technique for RDP, so we have used it as a correct answer: Brute Force (T1110) from the Credential Access tactic. We also mark the following as correct classification tactics: Discovery, Lateral Movement, Persistence, Impact, and Initial Access.
4. Dir listing—the Mitre database correspondence was as follows: File and Directory Discovery (T1083), Remote System Discovery (T1018), and the tactics Discovery, Reconnaissance, and Initial Access.

5. Cross-site scripting (XSS)—the Mitre database correspondence was as follows: Command and Scripting Interpreter (T1059) and the Execution tactic.
6. Port scanning—Mitre database correspondence was as follows: Network Service Discovery (T1046) and the Discovery tactic.

The detailed reports of individual experimental results are available in [8]. In this article, we have prepared a summary of the results based on a specific scoring method. The assigned score depends on two factors: correct technique (0.5 points) and tactic (0.5 points) classification. A score of 1.0 means a correct classification. If the model correctly found the attack technique but assigned a wider (but correct) tactic category, we have awarded only 0.25 points (e.g., for classifying something as Reconnaissance instead of Discovery). In some cases, the model was unable to respond in a timely manner due to the large amount of data in the constructed query. These cases were not classified (the score is represented by a dash). The final results are summarized in Table 4.

Table 4. Score obtained by the LLMs when classifying potential network attacks.

	Port Scan	RDP	DoS	Net Snif	XSS	Dir List	Score	Success
GPT-4	1	0	0.5	0	1	1	3.5	58%
GPT-3.5-turbo-0125	1	0	1	0.75	0.25	1	4	65%
Claude-3-haiku	0.75	1	1	0	1	1	4.75	80%
Mistral-7b-instruct	0.5	0.25	–	–	0.75	–	1.5	25%
Gemma-7b-it	0	0	–	0	0.75	–	0.75	12%

5.2. Cost-Benefit Analysis

The costs of our prototype solution depend on multiple factors. The main factor is the input size. That is, the number of tokens obtained from logs that the system needs to process. The other factor is the output size, which is the response that the model provides.

Prices for commercial models depend on the provider and subscription level. Table 5 summarizes the costs (per million tokens) valid during our testing.

Table 5. Prices of commercial models during our testing.

Model	Input Price [EUR per mil.]	Output Price [EUR per mil.]
GPT-3.5-turbo-0125	0.5	1.5
GPT-4	30	60
Claude-3-haiku	0.25	1.25

Model GPT-4 was the most expensive one, followed by GPT-3.5-turbo-0125 and Claude-3-haiku. When trying to classify 1 log event, we needed to send (on average) approximately 12,000 tokens twice to the model for each potential Suricata alert, obtaining answers of approximately 300 tokens (GPT-4 request cost was EUR 0.72 for input and 0.036 for output tokens). In the full network vulnerability testing phase, we have attained 480,000 processed tokens with 12,000 in response (GPT-4 total price of approx. EUR 15). A practical application would incur even higher costs due to the frequency of the security events. The usage of this application would thus be limited by economic factors instead of security considerations.

6. Discussion

We have presented two case studies that show potential applications of LLMs in the defensive cyber security domain. These two applications are similar in their work: they collect security-related information, prepare a prompt for LLM, and then process the results. The applications, however, have two different approaches to data collection and query construction. The first application (Android application penetration testing) is vulnerability-oriented: it pre-processes data based on the target vulnerability and only asks the model whether the processed data can represent the selected vulnerability. The second application (network monitoring) uses a more comprehensive approach: it collects all data relevant for some alert (potential incident), and then asks the LLM to identify whether it is (any) potential attack, and also wants the LLM to select a correct attack category.

The first system architecture was given by limitations of the context window. The amount of data related to a single application is too large to be presented to the LLM as a whole. These limitations were solved by creating specific Semgrep rules for specific vulnerability types and only presenting LLM with processed data. The success rate of the system is relatively low (including a large number of false positives) and precludes its use in automated security testing. The results, however, can be of use to human analysts to assist them in their work.

The second system used whole data related to specific identified Suricata alerts, although preprocessing was still needed to compress raw data into smaller data packets (using filters and aggregation). The system was incident-centric instead of category-centric unlike the previous one. This experiment also brought mixed results. When analyzing data from short time frames around potential incidents, the models were able to identify some of the attack techniques and tactics, but the success rate was also not high enough for an automated security response. The results became worse when we increased the time window to include more data. This was caused by noise introduced by other data unrelated to the actual attack. Furthermore, increasing the number of tokens in query led models to fail to answer.

We stress that the success rate of our tools is relatively low in comparison with dedicated AI-based cybersecurity tools (see also Section 6 for more details). The main cause of the low success rate is our decision to focus on existing pre-trained general language models. These models are not trained specifically for security tasks. Our objective was to show how successful these non-specialized models in specific security tasks are rather than solving these tasks with high accuracy.

We hypothesize that it would be possible to significantly improve the success rate by training specific models for dedicated security tasks (as other security research shows). This would, however, require significant investment and the models would be limited to the tasks they were trained on. An interesting research question is whether it is possible to efficiently combine pre-trained general models with dedicated smaller security models to achieve a better trade-off between general usability and task performance.

One of the limitations of the studied approaches is also the LLM context window size (see the original report [8] for details). Models can only process a limited number of tokens (and more tokens also incur higher prices). The context window limits the model's ability to find relevant data hidden in larger amounts of noise (such as in network logs or larger source codes of potentially vulnerable applications). In our experiments, the price of commercial models was also limiting, as we were significantly restricted in our tests.

Note that the context window size is not the only limitation of the LLMs in a security setting. For example, we have not addressed the questions of attacks targeting LLM-assisted detection and have not taken into account the detection of unknown (zero-day) vulnerabilities. We aimed to present two solutions where LLMs can assist human security analysts and report their success rate and limitations. The research in the AI area progresses at a rapid pace, and it is difficult to predict the future capabilities of LLMs.

From an economic point of view, the prices incurred by a single analysis range from fractions of a USD/EUR to tens of USD/EUR per scan/log analysis. On the other hand, the results of the analysis are not precise enough to completely remove a trained human operator. However, considering the average salaries of trained security analysts, AI-assisted analysis can be economically beneficial if the incurred costs are lower than the time-savings of the security personnel. To properly measure the benefits would, however, require a separate controlled study, which should compare the productivity of human security operators assisted by AI versus those that are unassisted.

Comparison with Related Works

In our paper, we focus on using LLMs in two concrete areas of cybersecurity in a setting where LLM is assisting a human analyst. Our current experience indicates that current LLMs are not yet suitable for the full automation of cybersecurity tasks due to their low reliability.

We point out that these results might significantly change in the future due to ongoing research efforts. We survey some more recent related works that solve similar problems with a different approach and can be used as a comparison to our results.

A novel vulnerability detection approach called GRACE was introduced in [20]. The GRACE approach introduces graph structure information into LLM for vulnerability detection, which can effectively address the limitations of LLM treating the code just as plain text. The GRACE approach could be integrated into our solution for the identification of Android vulnerabilities to improve the detection rate.

Another study that focuses on the improvement of the vulnerability detection approach is [21]. The paper aims to isolate LLMs' vulnerability reasoning from other capabilities, such as vulnerability knowledge adoption, context information retrieval, and structured output generation. They introduce a unified evaluation framework that separates and assesses LLMs' vulnerability reasoning capabilities and examines improvements when combined with other enhancements. Unlike other works (including our work), they were also able to identify new zero-day vulnerabilities. Zero-day vulnerability detection is generally not mentioned in conjunction with LLM approaches because of a need to create specific testing and benchmarking to prove that a zero-day vulnerability is detected, which is a difficult task.

The paper [22] investigates the effectiveness of LLMs in identifying vulnerabilities within codebases, with a focus on the latest advancements in LLM technology. They conclude that LLMs cannot be evaluated with just some known strong metric performances. An LLM excelling in a specific task does not automatically mean similar performance in related tasks. They believe that it is important to evaluate an LLM's performance in each task and field independently, similar to our approach.

Examples of other particular results and frameworks for vulnerability detection with LLMs in a specific cybersecurity domain include the following:

- Vul-RAG [23] focuses on building a vulnerability knowledge base by extracting multi-dimension knowledge via LLMs from existing CVE instances. Their vulnerability knowledge can improve manual detection accuracy.
- RealVul [24] is an LLM-based framework designed for PHP vulnerability detection. They employ various techniques to enhance detection proficiency: the identification of potential vulnerability triggers, the analysis of control flow and data flow, and the elimination of unnecessary semantic information.
- IRIS [25] combines LLMs with static analysis to perform whole-repository reasoning to detect security vulnerabilities. This approach is similar to ours, using CodeQL to extract metadata and then using LLMs in combination with a static analysis engine. They can detect 69 vulnerabilities from their dataset of 120 Java vulnerabilities while decreasing the number of false alarms in comparison with standard static analyzers.

Some studies focus specifically on Android vulnerabilities. In [26], a comparison of the ability of LLM models to detect Android code vulnerabilities is presented. The main difference between their approach and our study is that we do not compare detection abilities between SATS tools and LLMs, but we try to combine SATS with LLMs, and we investigate the possible improvement of this combination.

A comprehensive empirical study to explore the effectiveness of LLMs as test oracles for detecting Non-Crash Functional Bug (NCF bugs) in Android apps on 71 well-documented NCF bugs is presented in [27]. They achieve a 49% detection rate, similar to our results.

There are also many current works on the use of the LLMs in network cyber threat and attack detection. SecurityBERT [28] leverages the BERT model for cyber threat detection in IoT networks. They trained different types of cyber attack samples and focused on the detection, not the classification of threats. In our approach, the classification task is more relevant for the human analyst who uses the system.

A novel approach to interpreting the Intrusion Detection System (IDS) rules and predicting likely attacker Tactics, Techniques, and Procedures (TTPs) through the use of large language models (LLMs) is presented in [29]. Similar to our system, LLMs are trained to associate IDS rules to TTPs in the Mitre ATT&CK framework. Unlike our system, they train their model. While we obtain similar results with general LLMs, the paper shows that additional specialized training can improve the interpretability as well as the detection rate in some particular types of attacks.

7. Conclusions

In this paper, we present two case studies focusing on the potential use of LLM models in defensive cyber security. We have tested applications in the penetration testing of mobile applications as well as their use in the identification and classification of (network) security incidents. We have shown that LLMs have limited applications for automated solutions due to the imprecision of responses and incurred costs. They can, however, serve a human analyst as a support tool (along with other tools) to simplify the study of a large number of raw data involved in these tasks.

There are some interesting open research questions relevant to our study. Our data show that LLMs have a limited success rate in security-related tasks (in comparison to, for example, state-of-the-art malware detection), but it would be interesting to compare these results with human analysts with different expertise in given technical areas. Another research question is related to our approach to data compression. In the first approach, we used pre-processing related to a specific security vulnerability and provided LLM with data specifically for answering essentially a yes/no question about the presence of this type of vulnerability. In the second approach, we provided LLM with all data potentially relevant to some incidents and wanted to know whether the data indicated an attack and of what type. Our options for testing these approaches were limited, and it would be interesting to more precisely measure the impact of the representation on the success rate for the same type of data.

Author Contributions: Conceptualization, Š.B. and P.Z.; methodology, Š.B.; software, M.M. and O.V.; validation, Š.B., M.M. and O.V.; formal analysis, Š.B. and P.Z.; investigation, Š.B., M.M. and O.V.; resources, Š.B.; data curation, Š.B.; writing—original draft preparation, Š.B.; writing—review and editing, Š.B. and P.Z.; visualization, Š.B., M.M. and O.V.; supervision, P.Z.; project administration, P.Z.; funding acquisition, P.Z. All authors have read and agreed to the published version of the manuscript.

Funding: Š.B.: this work was supported by the Slovak Research and Development Agency under the Contract no. APVV-23-0292. P.Z.: this research was supported by Slovak Scientific Grant Agency grant VEGA 1/0105/23.

Data Availability Statement: Source code and supporting data can be found at https://github.com/StefanB172/cybersecurity-LLM (accessed on 26 November 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
CVE	Common Vulnerabilities and Exposures (database of known vulnerabilities)
LLM	Large Language Model (also: LLM-based tools, such as chatbots)
OWASP	Open Web Application Security Project
MobSF	Mobile Security Framework

References

1. Motlagh, F.N.; Hajizadeh, M.; Majd, M.; Najafi, P.; Cheng, F.; Meinel, C. Large language models in cybersecurity: State-of-the-art. *arXiv* **2024**, arXiv:2402.00891.
2. Divakaran, D.M.; Peddinti, S.T. LLMs for Cyber Security: New Opportunities. *arXiv* **2024**, arXiv:2404.11338.
3. da Silva, G.d.J.C.; Westphall, C.B. A Survey of Large Language Models in Cybersecurity. *arXiv* **2024**, arXiv:2402.16968.
4. Yao, Y.; Duan, J.; Xu, K.; Cai, Y.; Sun, Z.; Zhang, Y. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confid. Comput.* **2024**, *4*, 100211. [CrossRef]
5. Balogh, Š.; Mojžiš, J. New direction for malware detection using system features. In Proceedings of the 2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Metz, France, 18–21 September 2019; Volume 1, pp. 176–183.
6. Deng, G.; Liu, Y.; Mayoral-Vilches, V.; Liu, P.; Li, Y.; Xu, Y.; Zhang, T.; Liu, Y.; Pinzger, M.; Rass, S. Pentestgpt: An llm-empowered automatic penetration testing tool. *arXiv* **2023**, arXiv:2308.06782.
7. Mlynček, M. Using AI Against Attacks in Cyberspace. Master's Thesis, Slovak University of Technology in Bratislava, Bratislava, Slovakia, 2024. (In Slovak)
8. Vraňák, O. Confirmation of Incidents From IDS Logs Using AI. Master's Thesis, Slovak University of Technology in Bratislava, Bratislava, Slovakia, 2024. (In Slovak)
9. Karlsen, E.; Luo, X.; Zincir-Heywood, N.; Heywood, M. Benchmarking Large Language Models for Log Analysis, Security, and Interpretation. *J. Netw. Syst. Manag.* **2024**, *32*, 59. [CrossRef]
10. Li, H.; Shan, L. LLM-based Vulnerability Detection. In Proceedings of the 2023 International Conference on Human-Centered Cognitive Systems (HCCS), Cardiff, UK, 16–17 December 2023; pp. 1–4.
11. Tanksale, V. Cyber Threat Hunting Using Large Language Models. In Proceedings of the International Congress on Information and Communication Technology, London, UK, 19–22 February 2024; Springer: Singapore, 2024; pp. 629–641.
12. Piggott, B.; Patil, S.; Feng, G.; Odat, I.; Mukherjee, R.; Dharmalingam, B.; Liu, A. Net-GPT: A LLM-empowered man-in-the-middle chatbot for unmanned aerial vehicle. In Proceedings of the 2023 IEEE/ACM Symposium on Edge Computing (SEC), Wilmington, DE, USA, 6–9 December 2023; pp. 287–293.
13. Sandoval, G.; Pearce, H.; Nys, T.; Karri, R.; Garg, S.; Dolan-Gavitt, B. Lost at c: A user study on the security implications of large language model code assistants. In Proceedings of the 32nd USENIX Security Symposium (USENIX Security 23), Anaheim, CA, USA, 9–11 August 2023; pp. 2205–2222.
14. Zhang, J.; Wen, H.; Deng, L.; Xin, M.; Li, Z.; Li, L.; Zhu, H.; Sun, L. HackMentor: Fine-Tuning Large Language Models for Cybersecurity. In Proceedings of the 2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Exeter, UK, 1–3 November 2023; pp. 452–461.
15. Mohammed, S.P.; Hossain, G. Chatgpt in education, healthcare, and cybersecurity: Opportunities and challenges. In Proceedings of the 2024 IEEE 14th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 8–10 January 2024; pp. 0316–0321.
16. Tihanyi, N.; Ferrag, M.A.; Jain, R.; Debbah, M. Cybermetric: A benchmark dataset for evaluating large language models knowledge in cybersecurity. *arXiv* **2024**, arXiv:2402.07688.
17. Begou, N.; Vinoy, J.; Duda, A.; Korczyński, M. Exploring the dark side of AI: Advanced phishing attack design and deployment using chatgpt. In Proceedings of the 2023 IEEE Conference on Communications and Network Security (CNS), Orlando, FL, USA, 2–5 October 2023; pp. 1–6.
18. Scanlon, M.; Breitinger, F.; Hargreaves, C.; Hilgert, J.N.; Sheppard, J. ChatGPT for digital forensic investigation: The good, the bad, and the unknown. *Forensic Sci. Int. Digit. Investig.* **2023**, *46*, 301609. [CrossRef]
19. Zheng, S.; Zhang, Y.; Zhu, Y.; Xi, C.; Gao, P.; Zhou, X.; Chang, K.C.C. GPT-Fathom: Benchmarking Large Language Models to Decipher the Evolutionary Path towards GPT-4 and Beyond. *arXiv* **2023**, arXiv:2309.16583.
20. Lu, G.; Ju, X.; Chen, X.; Pei, W.; Cai, Z. GRACE: Empowering LLM-based software vulnerability detection with graph structure and in-context learning. *J. Syst. Softw.* **2024**, *212*, 112031. [CrossRef]
21. Sun, Y.; Wu, D.; Xue, Y.; Liu, H.; Ma, W.; Zhang, L.; Shi, M.; Liu, Y. LLM4Vuln: A Unified Evaluation Framework for Decoupling and Enhancing LLMs' Vulnerability Reasoning. *arXiv* **2024**, arXiv:2401.16185.
22. Sultana, S.; Afreen, S.; Eisty, N.U. Code Vulnerability Detection: A Comparative Analysis of Emerging Large Language Models. *arXiv* **2024**, arXiv:2409.10490.
23. Du, X.; Zheng, G.; Wang, K.; Feng, J.; Deng, W.; Liu, M.; Chen, B.; Peng, X.; Ma, T.; Lou, Y. Vul-RAG: Enhancing LLM-based Vulnerability Detection via Knowledge-level RAG. *arXiv* **2024**, arXiv:2406.11147.
24. Cao, D.; Liao, Y.; Shang, X. RealVul: Can We Detect Vulnerabilities in Web Applications with LLM? *arXiv* **2024**, arXiv:2410.07573.
25. Li, Z.; Dutta, S.; Naik, M. LLM-Assisted Static Analysis for Detecting Security Vulnerabilities. *arXiv* **2024**, arXiv:2405.17238.
26. Kouliaridis, V.; Karopoulos, G.; Kambourakis, G. Assessing the Effectiveness of LLMs in Android Application Vulnerability Analysis. *arXiv* **2024**, arXiv:2406.18894.
27. Ju, B.; Yang, J.; Yu, T.; Abdullayev, T.; Wu, Y.; Wang, D.; Zhao, Y. A Study of Using Multimodal LLMs for Non-Crash Functional Bug Detection in Android Apps. *arXiv* **2024**, arXiv:2407.19053.

28. Ferrag, M.A.; Ndhlovu, M.; Tihanyi, N.; Cordeiro, L.C.; Debbah, M.; Lestable, T.; Thandi, N.S. Revolutionizing cyber threat detection with large language models: A privacy-preserving bert-based lightweight model for IOT/IIOT devices. *IEEE Access* **2024**, *12*, 23733–23750. [CrossRef]
29. Gabrys, R.; Bilinski, M.; Fugate, S.; Silva, D. Using Natural Language Processing Tools to Infer Adversary Techniques and Tactics Under the Mitre ATT&CK Framework. In Proceedings of the 2024 IEEE 14th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 8–10 January 2024; pp. 0541–0547.