

## Article

# Intelligent Threat Detection—AI-Driven Analysis of Honeypot Data to Counter Cyber Threats

Phani Lanka <sup>\*</sup>, Khushi Gupta and Cihan Varol 

Department of Computer Science, Sam Houston State University, Huntsville, TX 77340, USA;  
kxg095@shsu.edu (K.G.); cxv007@shsu.edu (C.V.)

\* Correspondence: pklanka@shsu.edu or pklanka@gmail.com; Tel.: +1-4254353746

**Abstract:** Security adversaries are rampant on the Internet, constantly seeking vulnerabilities to exploit. The sheer proliferation of these sophisticated threats necessitates innovative and swift defensive measures to protect the vulnerable infrastructure. Tools such as honeypots effectively determine adversary behavior and safeguard critical organizational systems. However, it takes a significant amount of time to analyze these attacks on the honeypots, and by the time actionable intelligence is gathered from the attacker's tactics, techniques, and procedures (TTPs), it is often too late to prevent potential damage to the organization's critical systems. This paper contributes to the advancement of cybersecurity practices by presenting a cutting-edge methodology, capitalizing on the synergy between artificial intelligence and threat analysis to combat evolving cyber threats. The current research articulates a novel strategy, outlining a method to analyze large volumes of attacker data from honeypots utilizing large language models (LLMs) to assimilate TTPs and apply this knowledge to identify real-time anomalies in regular user activity. The effectiveness of this model is tested in real-world scenarios, demonstrating a notable reduction in response time for detecting malicious activities in critical infrastructure. Moreover, we delve into the proposed framework's practical implementation considerations and scalability, underscoring its adaptability in diverse organizational contexts.



**Citation:** Lanka, P.; Gupta, K.; Varol, C. Intelligent Threat Detection—AI-Driven Analysis of Honeypot Data to Counter Cyber Threats. *Electronics* **2024**, *13*, 2465. <https://doi.org/10.3390/electronics13132465>

Academic Editors: Salvador Otón Tortosa, Roberto Barchino, José Amelio Medina-Merodio and José Javier Martínez-Herráiz

Received: 22 May 2024

Revised: 15 June 2024

Accepted: 21 June 2024

Published: 24 June 2024



**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

**Keywords:** honeypots; computer security; cyberattack; data security; machine learning

## 1. Introduction

The escalating number of cybersecurity attacks on Internet-connected systems is a pressing concern. According to research published by the International Monetary Fund in 2024, cyberattacks have more than doubled since the pandemic [1]. The research also reveals a staggering increase in financial losses due to cybersecurity incidents, quadrupling since 2017 to \$2.5 billion. These figures underscore the urgency for security teams worldwide to detect and remediate attacks on their infrastructure promptly.

Unfortunately, data suggest that detecting a cyberattack is incredibly complex. Research published by IBM in October 2023 within the IBM Data Breach Action Guide indicates that it takes around 207 days on average to identify a cybersecurity breach and around 70 days to contain it once it is detected [2]. Meanwhile, there is a sharp reduction in the average amount of time it takes to launch an attack such as ransomware on enterprise networks. Research from IBM suggests that the time taken for a full-fledged ransomware attack has sharply fallen from 2 months to under 4 days. The pandemic has further complicated security threats in a typical office setting with the rise of remote work environments. The increased usage of cloud services and the trend of organizations exposing internal applications to the Internet using cloud platforms have significantly expanded the attack surface of typical corporate environments and associated infrastructure [3]. Both sophisticated Advanced Persistent Threats (APTs) and less sophisticated script kiddies are increasingly targeting cloud exposures [4]. The rise of sophisticated cyberattacks underscores the urgent

need for innovative detection mechanisms, such as User and Entity Behavior Analysis (UEBA), to confront the evolving threat landscape. UEBA systematically examines the typical behaviors of users and entities, utilizing this contextual understanding to pinpoint irregular deviations from established norms.

One of the mechanisms to target the adversaries attacking the infrastructure and understand their TTPs is by leveraging honeypots. A full-fledged high-interaction honeypot provides insights into attacker metadata, including the IP address they are coming from, the time of the attacks, the mechanism of compromise, and commands used to establish their persistence. Three types of honeypots offer different levels of glimpses into the adversary behavior [5,6]:

- Low-Interaction Honeypot: A low-interaction honeypot offers a plain vanilla TCP/IP service without access or with minimal access to the operating system. The adversary would not be able to interact with the honeypot, nor would the honeypots be capable of responding to the attackers in a way that captures their TTPs. The data yield from these honeypots is very low, but it could be easily analyzed to derive attacker information and use it to protect organizations' critical infrastructure. The quality of the data, especially the intention of attackers, cannot be captured in this type of honeypot [7];
- High-Interaction Honeypot: A high-interaction honeypot is on the other side of the spectrum compared to a low-interaction honeypot. This type of honeypot gives attackers complete root access to the operating system. Instead of merely mimicking specific protocols or services, the attacker is given authentic systems to target, significantly reducing the chances of detecting that they are being redirected or monitored [6]. A High-Interaction honeypot successfully captures the attackers' intentions and their TTPs. However, the data obtained from this type of honeypot are diverse, and the data analysis takes time to derive actionable insights for protecting organizations' critical infrastructure;
- Medium-Interaction Honeypot: A medium-interaction honeypot operates between a low-interaction and high-interaction honeypot, sacrificing some operating system authenticity to facilitate easier data analysis [6]. Organizations often resort to medium-interaction honeypots as a compromise, unable to rely on low-interaction honeypots due to their low data quality and unable to timely analyze the wealth of information offered by high-interaction honeypots. However, these honeypots are easily identified by adversaries, leading to a decline in the quality of extracted data.

Clouds are heavily targeted by attackers [8]. The more popular the cloud infrastructure, the more heavily the exposed infrastructure is targeted. Implementing honeypots in the cloud gives organizations a free glimpse into the attackers' activities, helping them prepare against malicious activity on the Internet.

One of the fundamental challenges in analyzing data from high-interaction honeypots is that several diverse shell commands can achieve an exactly similar outcome in an operating system. Blocking a specific shell command through either a block-list-based approach or an allow-list-based approach does not remotely do any justice in preventing attacks on the critical infrastructure of organizations. Many obfuscation techniques hide the actual command behind various shell-based masking techniques [9].

In the past couple of years, machine learning and artificial intelligence algorithms have grown exponentially. A *Bloomberg Intelligence* report states that AI-related spending among organizations will grow from 1% to 10% by 2032 [10]. The most significant revenue drivers will be generative AI infrastructure as a service used for training LLMs, digital ads, and specialized generative AI assistant software. One can use LLMs to detect malicious events in organizational critical systems. However, this notion comes with the following challenges:

- Most LLM models today are trained using historical datasets [11]. They can only evaluate a command if the training data references it. Even if the datasets being trained are current, the models do not have a mechanism today to pre-emptively understand the event and classify it as malicious;

- LLM models do not have enough information about attacks within an organization, especially the hosted honeypots, to identify and classify an event as malicious;
- It is also possible to train new LLMs with the organization's honeypot data. However, significant investments in cost and infrastructure are required to extract the relevant models while the honeypot data are still viable and usable from a security standpoint;
- Lastly, one way to derive a suitable classification of events in critical infrastructure (e.g., malicious, non-malicious) by an LLM is to provide data from the honeypot as a context for an LLM. However, the amount of data coming to the honeypot is significantly high. Providing this context might be impossible for the models where the amount of context exceeds the allowed number of tokens that can be used in a model. Further, as the context increases, the number of input tokens an LLM uses increases, making it cost-prohibitive and computationally expensive to leverage machine learning to identify malicious events.

Honeypots help derive precious data on attacker TTPs as they target a given organization. If there is a way to promptly convert these TTPs into actionable intelligence, the data's effectiveness and potential could be realized to prevent a malicious incident. With the advent of cloud computing, cyber adversaries are constantly probing for vulnerabilities in the open ports on the Internet [12]. Leveraging the advancements of machine learning algorithms and rich data from high-interaction honeypots, organizations can process data from high-interaction honeypots and detect incidents in their infrastructure as early as possible.

#### *Novelty of the Current Research*

Applying a machine learning model to any dataset would not result in spurious correlations. Machine learning must be applied to datasets where analysis of the data results in reasonable estimates of underlying patterns and relationships. Before analysis, it is paramount to ensure that the data are sufficiently clean, relevant, and representative of the security event classification domain. In essence, this research focuses on three research objectives:

- Develop a novel machine learning model tailored to the security event classification domain. The model aims to efficiently parse data from high-interaction honeypots and accurately identify malicious events in critical infrastructure;
- Investigate the effectiveness of leveraging native data analysis techniques with machine learning algorithms to score realistic events extracted from high-interaction honeypots. The aim is to enhance the accuracy and efficiency of identifying malicious events in critical infrastructure while reducing the time of such detection;
- Design and implement a comprehensive data pipeline integrating various analysis steps, including data parsing from high-interaction honeypots, machine learning-based event classification, and the application of the Retrieval and Augmented Generating (RAG) model to seek out and identify malicious events in critical infrastructure actively. Evaluate the performance and effectiveness of the proposed approach through real-world attack simulations in a cloud-based environment.

The document is arranged as follows: Section 2 presents some related work that has been leveraged to derive the model in this paper. Section 3 details the framework and explains the process of choosing the datasets for the current research, including a sample architecture used to validate the model. Section 4 discusses the results and observations identified through the established real-world attack scenario setup. Finally, Section 5 concludes this paper with key findings.

## 2. Related Work

Considerable machine learning research is being carried out across all domains, leveraging large language models (LLMs) and improvising with them to make them efficient and use the models for various datasets. Much of this research in the security space is focused on securing LLMs and preventing data exfiltration from LLMs or preventing the models from doing what they have been specifically instructed to do. However, very little research

has surfaced on integrating security data from log analysis, understanding attackers' TTP and mindset, and aligning it with an incident detection framework within an organization.

Research from No et al. [13] proposes a RAPID model leveraging log data's inherent characteristics for real-time anomaly detection without training delays. By treating logs as natural language and utilizing pre-trained language models for representation extraction, RAPID enables efficient detection without needing log-specific training, incorporating token-level information for refined and robust anomaly detection, particularly for unseen logs. The authors introduce the core set technique to reduce computational costs for comparison. Compared to prior models, experimental results demonstrate RAPID's competitive performance and effectiveness in real-time detection without delay. The research into the RAPID framework showcases how LLMs can be used to parse log files. Log statements are more human text readable compared to the bash commands executed on the honeypots. The key rationale behind using the RAPID model for the current research is to enhance the notion of using LLMs as a mechanism to analyze malicious commands from adversaries. The current research extends the notion of using LLMs to analyze malicious commands from adversaries and use them for anomaly detection.

Research from Karlsen et al. [14] examines the effectiveness of large language models (LLMs) in cybersecurity, particularly in analyzing log files. Various LLM architectures, including BERT, RoBERTa, DistilRoBERTa, GPT-2, and GPT-Neo, are benchmarked for their ability to analyze application and system log files for security purposes. The research deploys and benchmarks 60 fine-tuned language models for log analysis, demonstrating their effectiveness in this domain. Fine-tuning is found to be crucial for domain adaptation to specific log types. Additionally, the study introduces a new experimentation pipeline, LLM4Sec, which utilizes LLMs for log analysis experimentation, evaluation, and analysis. This research proves the versatility of using LLMs for analyzing logs. Key learning from this paper is that combining techniques enriching the context and augmenting LLMs with extra information would yield a robust detection mechanism. The current research leverages enhancements to the techniques proposed in the research from Karlsen et al. and improvises with them to analyze honeypot commands.

Research from Guu et al. [15] focuses on implementing a Retrieval and Augmented Generation model to enhance the capabilities of an LLM. LLMs store knowledge implicitly in network parameters, making it challenging to discern specific knowledge locations. REALM tackled this issue by introducing a discrete retrieval step called the 'textual knowledge retriever' in its pre-training algorithm. This retriever is incentivized to retrieve documents containing relevant information and penalized otherwise. REALM utilizes this mechanism to retrieve pertinent documents and focuses solely on these documents to make predictions, thereby enhancing the model's efficiency and effectiveness in leveraging stored knowledge. Retrieval and Augmented Generation models enhance the output of the LLMs. The current research leverages this model to parse the commands from the adversaries in the honeypot and query the commands executed by normal users in critical systems.

Research from Yang et al. [16] proposes adopting honeypot technology to shift from reactive to proactive cyber defense, aiming to address the limitations of typical defensive measures in current cyber confrontations. The system aims to enhance protective capabilities and ease of use by employing highly interactive honeypots and a modular design approach. The high-interactivity honeypot technology lures attackers into controlled environments for observation and performs advanced functions such as network threat analysis and vulnerability perception. It is thus possible to implement proactive detection measures leveraging data from the honeypot by effectively designing the honeypot and making the associated analysis modular and incremental. The research from Yang et al. showcases how high-interaction honeypots could be used for proactive defenses in a given organization. The current research leverages high-interaction honeypots to lure attackers and observe their behavior. Further, the modular framework proposed in the research by Yang et al. has been expanded to perform LLM analysis of the commands executed by the adversaries to aid in the detection of malicious commands in the critical infrastructure.

Research from Szabó and Bilicki [17] investigates using LLMs (specifically—GPT) for static analysis of front-end application source code to detect the CWE-653 vulnerability. By leveraging GPT's interpretive capabilities, the research aims to automate the identification of inadequately isolated sensitive code segments that could lead to unauthorized access or data leakage. Methodologically, the study involves classifying sensitive data, pre-processing code, and generating prompts for analysis using GPT. This paper showcases an initial framework for leveraging LLMs to interpret security issues that could be expanded to other use cases, especially log analysis. However, while this framework could be applied to programming languages, the honeypot logs are diverse and require more than prompt engineering to derive malicious events. The current research extends the research using Retrieval and Augmented Generation techniques through LLMs to automate the identification of malicious commands in the critical infrastructure leveraging analysis performed on the commands executed by the adversaries.

Research from Wang et al. [18] introduces AI@NTDS, a network threat detection system that uses behavioral features of attackers and intelligent techniques. It combines data analysis, feature extraction, and evaluation to build a detection model, aiding operating systems in defending against network attacks. Linux system interaction data from SSH and Telnet are collected from Cowrie Honeypot and labeled based on MITRE ATT&CK tactics for dataset credibility. The key learning for the research from Wang et al. is that SSH and Telnet provide a solid detection model to identify malicious behavior in the honeypots. However, Cowrie is still a medium-interaction honeypot. The current research leverages a high-interaction honeypot and more complex machine learning models and LLMs to derive the adversary behavior and compare it with user commands in the critical infrastructure.

Research from Lanka et al. [19] verifies that security adversaries exploit various targets, focusing on easy compromises to extend their attacks. Cloud environments, including AWS, Azure, GCP, and OCI, are prime targets due to the volume of attacks, offering insights into attacker objectives and patterns. The study examines adversary practices on commonly exposed protocols in these platforms, documenting a honeypot model that compares attacker behavior across multiple cloud environments. Additionally, the article highlights security measures to mitigate threats from adversaries probing insecure targets on cloud platforms. The model from this paper helps drive the honeypot design and validate the machine learning model in the current research. The research from Lanka et al. stops short of utilizing adversary behavior to identify malicious commands from user interactions with critical infrastructure. The current research expands into this notion and showcases a model through which the commands executed in the honeypots could be used to analyze the commands executed in the critical infrastructure.

Research from Lewis et al. [20] investigates the limitations of large pre-trained language models in accessing and manipulating knowledge, which impacts their performance on knowledge-intensive tasks. The authors propose a general-purpose fine-tuning approach for Retrieval and Augmented Generation (RAG) models, which integrate pre-trained parametric and non-parametric memory for language generation. The study compares two RAG formulations: one that uses the same retrieved passages for the entire generated sequence and another that allows different passages per token. For language generation tasks, RAG models produce more specific, diverse, and factual language compared to a state-of-the-art parametric-only seq2seq baseline. RAG can be used in the current research to compare commands. The commands may vary significantly between adversaries, but the essence of command comparison lies in using the description of the commands and comparing them against each other.

### 3. Methodology

Table 1 illustrates a sample log event from an SSH-based high-interaction honeypot and their corresponding timestamps. Typical honeypot logs contain extensive information about the interaction with the adversary. For instance, in the session depicted in Table 1, the following could be deduced from the sample attack that occurred over just 22 s:

- Attacker's IP address;
- Attacker credentials;
- Protocol data;
- Timestamps of each event;
- Adversary commands.

**Table 1.** Illustrated SSH honeypot logs.

SSH Message Type	Source	Destination	Payload
Connect	Client	Server	{RemoteAddr: " 183.81.169.238 ", Country: " HK "} Timestamp: 1715108300
Password authentication	Client	Server	Username: "root", Password: "0" Timestamp: 1715108300
Password authentication successful	Server	Client	Username: "root", Authenticated: "true" Timestamp: 1715108302
New channel successful	Server	Client	New channel created Timestamp: 1715108302
Execute program	Client	Server	Program: apt update && apt install sudo curl -y && sudo useradd -m -p \$(openssl passwd -1 DPdMYqVR) system && sudo usermod -aG sudo system Timestamp: 1715108302
SSH I/O	Server	Client	Program output: WARNING: apt does not have a stable CLI interface. Use with caution in scripts.\n... Timestamp: 1715108310
Execute program	Client	Server	Program: lscpu && echo -e DPdMYqVR\nDPdMYqVR passwd && curl https://ipinfo.io/org --insecure -s && free -h && apt Timestamp: 1715108310
SSH I/O	Server	Client	Message: Architecture: x86_64\n <ul style="list-style-type: none"> <li>CPU op-mode(s): 32-bit, 64-bit\n</li> <li>Address sizes: 48 bits physical, 48 bits virtual\n</li> <li>Byte Order: Little Endian\n</li> <li>CPU(s): 2\n</li> <li>On-line CPU(s) list: 0,1\n</li> </ul> ... Timestamp: 1715108322
Disconnect	Client	Server	Client Disconnect Timestamp: 1715108322

Legend: (1) Green labeled text illustrates the timestamps, (2) blue labeled text indicates SSH data transfer, and (3) yellow labeled text indicates protocol parsed value.

Although Table 1 depicts a sample log involving the SSH protocol, similar information can be deduced from honeypots of any protocol. High-interaction honeypots yield large

volumes of discrete data from various interactions. The datasets are not similar to one another. Therefore, significant processing is required before deriving meaningful outcomes from the data and using it for threat detection in organizations.

The first pre-processing step involves identifying the types of datasets from the logs. Specifically, it involves identifying those datasets that could be used for active learning and those that could indicate a compromise without any machine learning analysis. Table 2 represents further information that could be deduced from the datasets identified in Table 1. Each subsection below details the relevance of these data elements in analyzing malicious commands and discusses how they could be refined further to aid in the identification of malicious activity in the critical infrastructure.

**Table 2.** Data types collected from honeypot.

Data Field	Type of Data	Fit for Machine Learning	Explanation/Behavior
Attacker IP Address	Identification Data	No	IP addresses in honeypots are used to identify malicious actors from the Internet.
Attacker Credentials	Identification Data	No	Credentials from attackers usually represent the dataset these adversaries use to perform brute force attacks and do not provide any benefit for learning.
Connection Creation Attempts	Protocol Data	No	Different types of protocols offer different methodologies for connection creation. Connection creation attempts are also influenced by how stable the underlying connection to the honeypot is. These data would not provide any benefit for learning.
Commands Executed	Attacker Commands	Yes	Commands executed by the attacker would provide a glimpse into their TTPs. Learning from these data would aid in malicious activity detection.
Total Execution Time	Activity Data/Numeric Data	Yes	Attackers are on a constant lookout for externally exposed services and strongly focus on establishing persistence in the cloud [19]. These data can be used for machine learning classification to identify malicious activity.

### 3.1. Attacker IP Address

Attacker identity-based datasets, particularly IP addresses and connection credentials, are considered direct indicators of compromise. The IP addresses cannot be correlated through any analysis or machine learning methodologies. In essence, these are the types of strings that, even when learned using a machine learning model, the predictions from the model would not yield results that could be used to detect malicious activity on the company's sensitive infrastructure. However, these strings could be used independently to identify suspicious activity. For instance, the activity could be considered suspicious if the attacks happened on the honeypot from an IP address 183.81.169.238 and new connections on critical infrastructure originated from that IP address in a relatively short time (e.g., past week).

### 3.2. Attacker Credentials

Similarly, if the attacker uses a password to log into both the honeypot and critical infrastructure, this increases the suspicion that the activity in critical infrastructure is malicious. Usage of same credentials in both honeypot and critical infrastructure suggests a deliberate effort by the adversary to penetrate secure systems, signaling malicious intent rather than incidental or innocuous actions. In addition, identification of identical pass-

words raises alarms about the potential for additional infiltration or harm elsewhere within the critical infrastructure, which previously exposed user credentials to the attacker. This scenario underscores the importance of robust cybersecurity protocols and continuous monitoring to promptly detect and respond to malicious behavior. Additionally, it highlights the significance of threat intelligence and collaborative information sharing in identifying and countering the strategies and techniques employed by attackers, empowering organizations to enhance their defenses against future attacks.

Sometimes, user credentials are weak and do not adhere to secure password generation methodologies. An attacker can guess the password of legitimate users without compromising any other service in the organization instead of making a deliberate effort to compromise credentials. These types of attacks are easy for attackers and come at no cost, leading to increased attacks in cloud environments [19]. Regardless of the underlying root cause, attackers using identical credentials for both honeypots and critical infrastructure pose a significant danger to an organization, as it grants adversaries nearly resistance-free access to the infrastructure.

### 3.3. Protocol Data

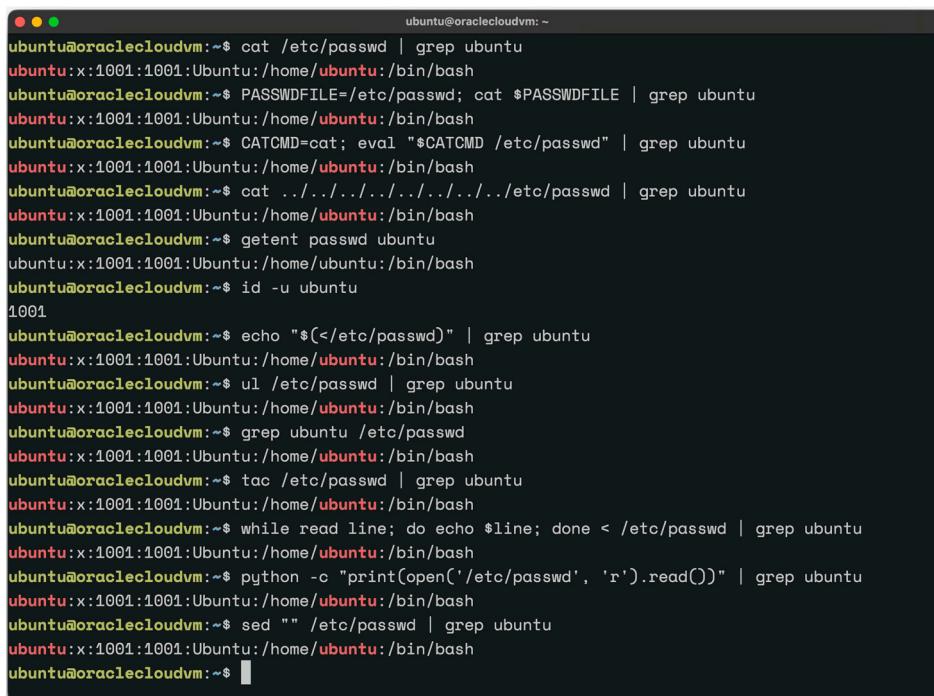
Protocol data indicates the behavior of a connection. Many connection attempts to the server in a short timeframe and increase in number of channel requests usually indicate poor connectivity to the honeypot regardless of the protocol. Since protocol data cannot be directly quantified in all cases, it usually does not yield any helpful metric, either by direct correlation or machine learning, to identify adverse security events.

### 3.4. Timestamp Data

Timestamps usually indicate attackers' active timeframes. However, with attacks on exposed ports happening around the clock, especially in the cloud, and adversaries following automation procedures, the attack timeframe does not provide a direct mechanism to learn and identify malicious threats to sensitive infrastructure. However, these timestamps can be used to determine the age of an attack and its relevance when parsing events from critical infrastructure. Furthermore, the timestamps could determine the attacker's engagement time in a session. Automated sessions by adversaries typically take less time to execute a sequence of commands than manual/human sessions without automation. The difference in session times between honeypot and critical infrastructure and the number of command-based issues could be used to identify the probability of malicious activity.

### 3.5. Adversary Commands

Commands executed by the attacker directly translate into adversaries' TTPs and can provide valuable information for identifying malicious events in critical infrastructure. However, adversaries can alter these commands to deduce similar information from the servers. These commands can sometimes be obfuscated or unnecessarily chained to avoid detection. The commands yield a wealth of information; however, reading them as strings would not help since some of the commands are variable even though the underlying TTPs of attackers are consistent. As an example, Figure 1 demonstrates a non-exhaustive example list of commands, each one checking if the user "ubuntu" exists in the system. Therefore, some retrieval is needed to ensure these commands are normalized to derive the attacker's TTP.



```

ubuntu@oraclecloudvm:~$ cat /etc/passwd | grep ubuntu
ubuntu:x:1001:1001:Ubuntu:/home/ubuntu:/bin/bash
ubuntu@oraclecloudvm:~$ PASSWDFILE=/etc/passwd; cat $PASSWDFILE | grep ubuntu
ubuntu:x:1001:1001:Ubuntu:/home/ubuntu:/bin/bash
ubuntu@oraclecloudvm:~$ CATCMD=cat; eval "$CATCMD /etc/passwd" | grep ubuntu
ubuntu:x:1001:1001:Ubuntu:/home/ubuntu:/bin/bash
ubuntu@oraclecloudvm:~$ cat ../../../../../../etc/passwd | grep ubuntu
ubuntu:x:1001:1001:Ubuntu:/home/ubuntu:/bin/bash
ubuntu@oraclecloudvm:~$ getent passwd ubuntu
ubuntu:x:1001:1001:Ubuntu:/home/ubuntu:/bin/bash
ubuntu@oraclecloudvm:~$ id -u ubuntu
1001
ubuntu@oraclecloudvm:~$ echo "$(cat /etc/passwd)" | grep ubuntu
ubuntu:x:1001:1001:Ubuntu:/home/ubuntu:/bin/bash
ubuntu@oraclecloudvm:~$ ul /etc/passwd | grep ubuntu
ubuntu:x:1001:1001:Ubuntu:/home/ubuntu:/bin/bash
ubuntu@oraclecloudvm:~$ grep ubuntu /etc/passwd
ubuntu:x:1001:1001:Ubuntu:/home/ubuntu:/bin/bash
ubuntu@oraclecloudvm:~$ tac /etc/passwd | grep ubuntu
ubuntu:x:1001:1001:Ubuntu:/home/ubuntu:/bin/bash
ubuntu@oraclecloudvm:~$ while read line; do echo $line; done < /etc/passwd | grep ubuntu
ubuntu:x:1001:1001:Ubuntu:/home/ubuntu:/bin/bash
ubuntu@oraclecloudvm:~$ python -c "print(open('/etc/passwd', 'r').read())" | grep ubuntu
ubuntu:x:1001:1001:Ubuntu:/home/ubuntu:/bin/bash
ubuntu@oraclecloudvm:~$ sed "" /etc/passwd | grep ubuntu
ubuntu:x:1001:1001:Ubuntu:/home/ubuntu:/bin/bash
ubuntu@oraclecloudvm:~$ █

```

**Figure 1.** Sample list of commands to identify if user “ubuntu” exists in bash shell.

### 3.5.1. Command Retrieval Using LLM

Due to the varied nature of commands with similar TTPs, treating the commands as strings is not the correct approach to analyzing adversary behavior. However, LLMs can be leveraged to analyze the command and understand the attackers’ underlying TTP. Appropriate prompts must be used to ensure that LLMs yield a predictable output each time a command is executed. Research from Amatriain explores creating custom prompts that deliver quality results from LLM models [21]. Out of all the methodologies used to retrieve results from an LLM, Chain of Thought (CoT) prompting, Role Playing, Teaching Algorithm in Prompt, and Generate Different Opinions deliver a robust output for the datasets parsed in this work.

Table 3 provides information on the prompt used in this research to parse adversary commands and derive the attacker’s TTPs. The instructions and prompts will be executed in the model “gpt-4-turbo” in May 2024. Table 4 represents the sample output by LLM after parsing the commands listed in Figure 1. As shown, the LLM can parse the commands and present clean output that clearly articulates the attacker’s TTP, regardless of how different a command is that is executed in the honeypot.

However, the challenge with LLM-generated text is that it still contains variables that are inputted into the command. For instance, the username “ubuntu” is a variable sent to the bash command, and the attacker TTPs are unaffected even if the variable value changes to a different username (e.g., “johndoe”). Therefore, LLM-generated data cannot be directly used for text-based matching. Certain post-processing is required for LLM-generated data before it is stored in the database for later searches. Bashlex library provides a Python-based bash parsing library to interpret a complex bash command and identify the words/arguments provided [22]. Using Bashlex helps identify the arguments/entities in the bash commands issued by the adversary. Table 5 shows a sample list of commands executed in the vulnerable honeypot and the identification of entities from the data. The identified entities are replaced with generic strings in the LLM responses in Table 4.

**Table 3.** Prompts used to derive adversary TTPs.

Technique Used	Prompt Additional Instruction
Role Playing	You are an advanced Linux bash interpreter. You will be given explicit commands that are executed in a bash shell in a honeypot server.
Setting Expectations for Input	These bash shell commands were executed by malicious users. All the bash commands must be assumed to be malicious.
Setting Expectations for Input	You would need to interpret the malicious bash commands and identify the tactics used by the malicious user
Setting Expectations for Output	The output of your analysis must be list of bullet points that has verbose description of what bash commands are doing.
Setting Expectations for Output	The output must be in plain English, showing details of what command does without any reference to the command or its arguments.
Teaching Algorithm in Prompt	For example, in cases of sample input “ls/”, the output must be “This command lists the directories and files in the filesystem root of the server in a non-recursive manner. The default implementation of the command does not show the hidden directories and files in the server”.
Chain of Thought (CoT)	Take time before you respond. Please proceed step by step and think this through thoroughly for accuracy.
Generate Different Options	Please explore multiple potential interpretations and implications of each command to ensure a comprehensive and accurate analysis.
Setting Expectations for Output	Please provide no greetings or offer help toward the end. Please provide the responses in bullet list of sentences.

Finally, once the LLM responses have been replaced with generic entities, the resultant LLM text is converted into an embedding vector for efficient searching during the augmentation step. Multiple embedding models can transform the text into vectors, and this area has been the focus of research to identify more efficient models. For the current research, “Salesforce/SFR-Embedding-Mistral” is used as the embedding model to parse the text since this is one of the top models for retrieval in the Hugging Face MTEB as of May 2024 [23,24]. Since this model is quite large, executing it requires an A100 GPU running in a Google Colab environment. However, augmentation can also be achieved with smaller models in a local environment. These embeddings generated can be stored in database and can be searched to match the TTPs from critical infrastructure during the retrieval process.

The retrieval part of the model utilizes commands generated by the company’s critical infrastructure users. To ensure consistency with the embeddings, the processing for user commands must adhere to the same processing steps as before. The user command is forwarded to an LLM with an appropriate prompt to retrieve the command description in a format consistent with honeypot models. After obtaining the description, the embeddings are computed using the “Salesforce/SFR-Embedding-Mistral” as the embedding model to analyze the text. The model Salesforce/SFR-Embedding-Mistral requires the query to be of a specific task description format to be input with the query. Figure 2 shows a sample task description and response from the model.

**Table 4.** Analysis of sample adversary commands in Figure 1 using the LLM “gpt-4-turbo”.

Command Used	LLM Output Using Prompt in Table 3
<code>cat/etc/passwd   grep ubuntu</code>	<ul style="list-style-type: none"> <li>This command retrieves the content of the server’s user database file, which includes details like user IDs, user names, home directories, and shell information;</li> <li>It then filters through this content, specifically looking for occurrences of the string “ubuntu” to find information related to a user or users with the username containing “ubuntu”;</li> <li>The primary function is to extract specific user account information to analyze or modify user properties or to verify the presence and details of certain user accounts.</li> </ul>
<code>PASSWDFILE = /etc/passwd; cat \$PASSWDFILE   grep ubuntu</code>	<ul style="list-style-type: none"> <li>This sequence of commands sets an environment variable named PASSWDFILE with the path to the server’s user database file;</li> <li>After setting the variable, it displays the content of the file specified by PASSWDFILE and filters this content to look for occurrences of the string “ubuntu”;</li> <li>The primary objective is to extract and display specific information about user accounts that include “ubuntu” in their details, which can include usernames, home directories, or shell paths. This action might be used to inspect or manipulate user properties or to validate the existence and specifics of particular user accounts.</li> </ul>
<code>CATCMD = cat; eval "\$CATCMD/etc/passwd"   grep ubuntu</code>	<ul style="list-style-type: none"> <li>This command sequence sets an environment variable CATCMD to store the name of a Unix command used for displaying the content of files;</li> <li>Using eval, it executes the command stored in CATCMD to display the content of the server’s user database file;</li> <li>The output of this command is then filtered to specifically search for entries containing the term “ubuntu”;</li> <li>The purpose is essentially to isolate and reveal details pertaining to users with “ubuntu” in their information from the user database. This can be used for inspecting user configurations, checking user existence, or analyzing security policies related to those user accounts.</li> </ul>
<code>cat../../../../../../../../etc/passwd   grep ubuntu</code>	<ul style="list-style-type: none"> <li>This command sequence displays the contents of the user database file from the absolute root directory by repeatedly traversing up through parent directories until the root is reached and then accessing the/etc/passwd file;</li> <li>It then filters the displayed content to find lines containing the text “ubuntu”, typically to identify information regarding any user whose details include “ubuntu” in the user database;</li> <li>The primary aim is to search for and isolate specific user information potentially for unauthorized access, exploitation, or verification of specific user existence from a sensitive system file that stores all user details.</li> </ul>
<code>echo "\$(</code> </etc/passwd <code>)"   grep ubuntu</code>	<ul style="list-style-type: none"> <li>This command first reads the entire contents of the server’s user database file directly into memory without explicitly calling a separate command to display the file;</li> <li>This input is processed to specifically filter and show details that involve the term “ubuntu”, likely echoing back information about a user or users related to “ubuntu”;</li> <li>The primary intent of this operation is to precisely extract user-specific information in order to monitor, verify, or manipulate details regarding user accounts on the system, specifically those involving ‘ubuntu’.</li> </ul>

**Table 5.** Identification of command line attributes (entities) from bash command.

Command Used	Identified Entities
<code>cat/etc/passwd   grep ubuntu</code>	"/etc/passwd", "ubuntu"
<code>PASSWDFILE = /etc/passwd; cat \$PASSWDFILE   grep ubuntu</code>	"/etc/passwd", "ubuntu"
<code>ps aux   grep ld-musl-x86   grep -v grep   grep -v systemd-networkd   grep -v ld-musl-x86_64   grep -v rsyslogd   wc -l</code>	"ld-musl-x86", "systemd-networkd", "ld-musl-x86_64", "rsyslogd"
<code>kill -9 \$(ps aux   grep zzh   grep -v grep   awk '{print \$2}')</code>	"zzh"
<code>scp -t/tmp/iy6ii01nig469bn3a5ce66a8t4</code>	"iy6ii01nig469bn3a5ce66a8t4"
<code>echo -e \"C9QXng3P\\nC9QXng3P\"   passwd</code>	"C9QXng3P"
<code>curl https://ipinfo.io/org --insecure -s</code>	"ipinfo.io", "org"

```

from sentence_transformers import SentenceTransformer
from tqdm import tqdm
import random

embedding_model = SentenceTransformer(model_name_or_path="Salesforce/SFR-Embedding-Mistral",
                                         device="cuda")

for item in tqdm(chunks):
    item["embedding"] = embedding_model.encode(item["joined_sentence_chunk"])

random.choice(chunks)

Download shards: 100% [██████████] 3/3 [00:42<00:00, 14.31s/it]
model-00001-of-00003.safetensors: 100% [██████████] 4.94G/4.94G [00:12<00:00, 279MB/s]
model-00002-of-00003.safetensors: 100% [██████████] 5.00G/5.00G [00:14<00:00, 390MB/s]
model-00003-of-00003.safetensors: 100% [██████████] 4.28G/4.28G [00:13<00:00, 234MB/s]
Loading checkpoint shards: 100% [██████████] 3/3 [00:04<00:00, 1.39s/it]
tokenizer_config.json: 100% [██████████] 981/981 [00:00<00:00, 87.8kB/s]
tokenizer.model: 100% [██████████] 493k/493k [00:00<00:00, 36.5MB/s]
tokenizer.json: 100% [██████████] 1.80M/1.80M [00:00<00:00, 7.88MB/s]
special_tokens_map.json: 100% [██████████] 624/624 [00:00<00:00, 48.3kB/s]
1_Pooling/config.json: 100% [██████████] 271/271 [00:00<00:00, 22.2kB/s]
100% [██████████] 125/125 [00:45<00:00, 2.74it/s]
{'command': 'ps aux | grep systemd-net | grep -v grep | grep -v systemd-networkd | grep -v ld-musl-x86_64 | grep -v rsyslogd | wc -l',
 'explanation': '- The command sequence provided aims to count the number of processes related to `systemd-net` with specific exclusions. Here is a detailed explanation of each part:\n1. `ps aux`: This lists all the running processes on the system with detailed information about each process.\n2. `grep systemd-net`: Excludes any lines that contain the string `grep`.\n3. `grep -v grep`: Excludes any lines that contain the string `grep`, which generally appear because the `grep` command searching for `systemd-net` shows up in the process list.\n4. `grep -v systemd-networkd`: Excludes lines that contain `systemd-networkd` to refine the search further to exclude the main system network service daemon.\n5. `grep -v ld-musl-x86_64`: Excludes any'

```

**Figure 2.** Execution of “Salesforce/SFR-Embedding-Mistral”.

Embeddings from the model for the query are matched against embeddings from the honeypot commands. The embeddings from “Salesforce/SFR-Embedding-Mistral” are not normalized. Hence, cosine similarity is used to obtain the matching vectors from the embeddings derived from the command. The cosine similarity metric facilitates vector comparison by quantifying the cosine of the angle between them, offering a robust measure of directional similarity while disregarding magnitude discrepancies. Its application transcends various domains, enabling precise analysis and classification tasks in complex datasets. The cosine similarity function provides two outputs: the similarity score that illustrates the level of match between the texts and the list of indices that match the score.

The retrieval mechanism aims to match the top ‘k’ results against the input value. The value of this threshold ‘k’ depends on multiple factors, such as the sensitivity of the critical infrastructure, the underlying model used for similarity analysis, the organization’s appetite for the false positives and the false negatives, and the similarity between honeypot infrastructure and sensitive critical infrastructure. It is important to note that a lower value of this threshold increases the false positives if the adversary’s commands differ from those executed in the sensitive infrastructure. In comparison, a higher value of the threshold increases false negatives and LLM hallucinations. Also, an increase in the value of ‘k’ would further increase the number of tokens in the input, thus increasing the underlying cost of

performing LLM searches. In the current scenario of a simple honeypot and a mimicking SSH server, the ‘k’ value selected is 3. However, this value could be varied depending on the criteria listed above.

### 3.5.2. Augmentation Using LLM

A retrieval analysis of honeypot commands from an adversary typically yields a reasonably good measure of similarity to the commands executed in the sensitive infrastructure. The magnitude of cosine similarity scores indicates the extent of match between two commands. A cosine similarity score of values closer to 1.0 means a closer match between the commands executed in critical infrastructure and the honeypot. The objective of the current research is to identify malicious events in the critical infrastructure. Therefore, a specific value threshold in the cosine similarity score must be considered to ensure that only similar values are selected for subsequent analysis.

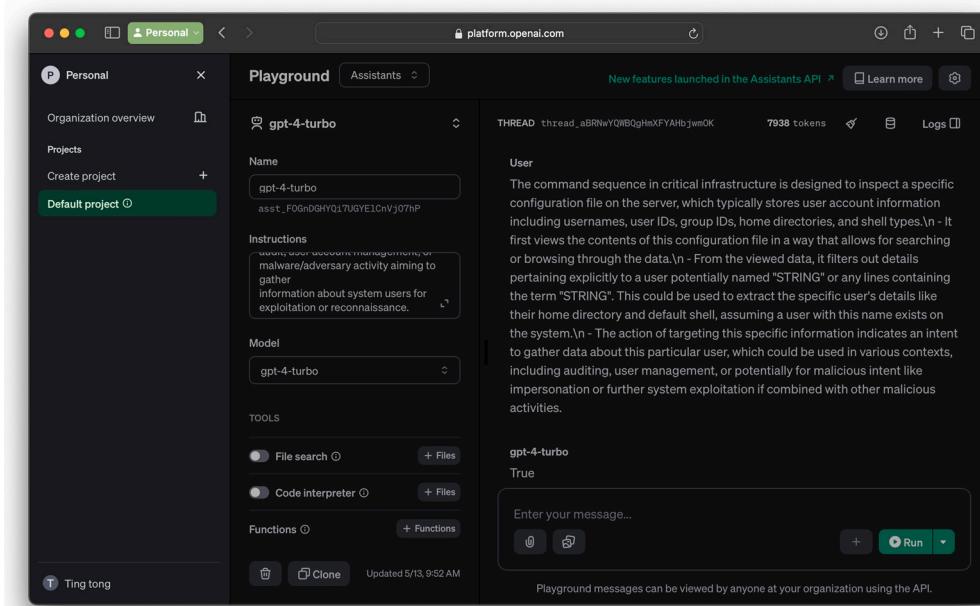
To ensure research objectives are met, it is not necessary to consider every command that exceeds the cosine similarity threshold for our LLM analysis. Instead, focus on the top ‘k’ values from the cosine similarity results that exceed the selected threshold. An LLM can be used to analyze these top ‘k’ search results, allowing it to quantitatively determine whether a command is malicious or benign and the category of the attacker TTPs employed in the command. By employing suitable prompt engineering approaches and providing context from the retrieved queries, the LLM can be augmented to generate quantifiable feedback. Table 6 lists the instructions given to “gpt-4-turbo” for generating responses to the query. This query uses a Role Playing and Teaching Algorithm in the prompt to assess if the command is malicious. The query also ensures the LLM interprets the input and provides a minimal binary output for further analysis. Further, LLMs can also derive the attack category the adversary uses. MITRE publishes a global database of enterprise ATT&CK categories and knowledge base [25]. Leveraging the top attack categories from the list, Table 6 lists the query to determine the category of the attackers’ TTP.

**Table 6.** Prompts used to derive responses to infrastructure commands.

Query	Technique Used	Prompt Additional Instruction
Determine if the user interaction in critical interaction is malicious	Role Playing	You are a security monitoring expert
	Setting Expectations for Input	You would be given a bulleted list of command descriptions that adversaries have used against our organization’s honeypot. These are called attacker TTPs
	Setting Expectations for Output	For a given query, you need to provide output that indicates whether the query command description matches any of the attacker’s TTPs
	Teaching Algorithm in Prompt	For example, if the query description matches the attacker TTPs in the description, return with value of True. Otherwise, return with value of False
	Setting Expectations for Output	Please provide no greetings or offer help toward the end. Please provide the responses in either True or False
Determine the ATT&CK category	Role Playing	You are a security monitoring expert
	Setting Expectations for Input	You would be given a bulleted list of command descriptions that adversaries have used against our organization’s honeypot. These are called attacker TTPs
	Setting Expectations for Output	For a given query, leverage the provided TTPs and categorize the query into one of the ATT&CK categories—“Reconnaissance”, “Persistence”, “Impact”, “Exfiltration”, and “Command and Control”
	Setting Expectations for Output	Please provide no greetings or offer help toward the end. Please provide the responses as a single word with category name

### 3.5.3. Generation Using LLM

Figure 3 provides a snapshot of this LLM prompt in action. Leveraging the query and the instructions, an LLM could be used to determine if the command executed in the critical infrastructure is malicious. As part of the research, “gpt-4-turbo” LLM is used for the analysis. However, this could be easily replaced with lower-cost LLMs and those that run natively within the organization’s infrastructure. A combination of attacker categories, attack command matches, and command session time learning could be used to derive if a given event in critical infrastructure is malicious.



**Figure 3.** Sample response to user command query to identify malicious transaction.

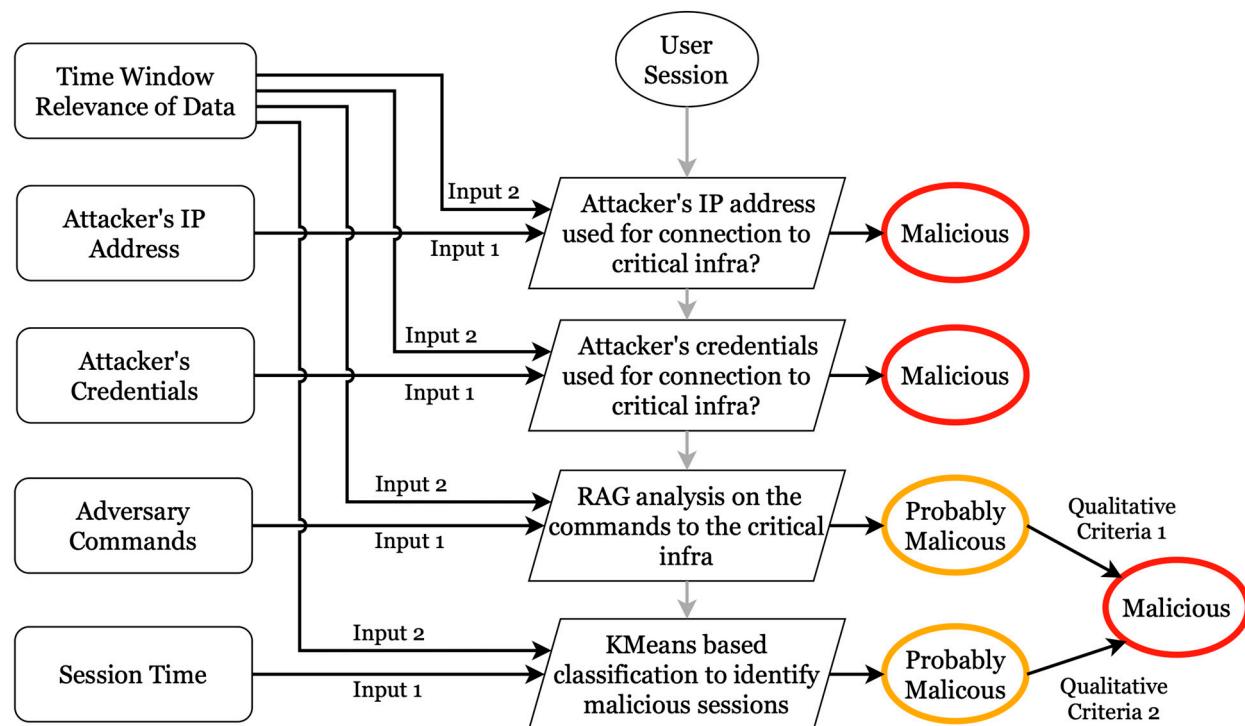
### 3.6. Putting Model Together—Creating a Pipeline

While the example focuses on SSH, similar insights into adversaries can be extracted from other protocol-based honeypots. As indicated in the sections above, the datasets must be pre-processed due to the diverse nature of datasets gathered from high-interaction honeypots, which require careful consolidation before integrating them into machine learning frameworks. Although attacker IP addresses and credentials are not directly conducive to machine learning and data analysis, they can indicate suspicious activity when cross-referenced across honeypots and infrastructure logs. The protocol and timestamp data present challenges in directly quantifying malicious threats. Adversary commands provide valuable insights into tactics, techniques, and procedures (TTPs) and can be analyzed using language model-based techniques like LLMs. Utilizing LLMs, specifically Retrieval and Augmented Generation (RAG), enables TTP extraction from commands, aiding in identifying malicious activity. These methodologies, coupled with suitable embedding models and retrieval mechanisms, facilitate comprehensive analysis and identification of threats within critical infrastructure, bolstering cybersecurity defenses.

#### 3.6.1. Model Algorithm Implementation

Figure 4 describes the overall algorithm for the model. The model is constructed through a series of if–then–else statements that associate a given command executed in critical infrastructure as malicious or benign. Before a model can be constructed, it is essential to identify a sufficient time window based on which the rest of the analysis can be associated. The time window is important because the commands could be classified as malicious or benign within the specific time window based on the attacks happening concurrently in the honeypot. It would not be ideal to define an event in critical infrastruc-

ture as malicious, referencing a honeypot event that occurred months or even years ago. The time window length depends on various factors, such as the organization industry, the compliance window needed for the reference time frame, the nature of the endpoint, and the amount of traffic received to the honeypot. For this paper, the time window of the analysis is selected to be a week (7 days).



**Figure 4.** Flow chart of the proposed model.

For a given time window, various user parameters to the critical infrastructure are iteratively parsed, as described in Figure 4, to determine if a given command is malicious. The attacker's IP address and credentials directly correlate with the malicious activity. If these are identified in the critical infrastructure, they must be immediately classified as malicious. Machine learning algorithms are executed on each of the remaining datasets—attacker commands executed and attacker session time. RAG analysis is performed on the attacker commands, resulting in a True or False result along with TTP category result, all together, indicating whether the command is suspicious or benign. Attacker session time also provides a mechanism to classify the transaction. Algorithms such as the K-Means could be applied to classify existing benign sessions in the critical infrastructure and adversary sessions in the honeypot in the time frame selected. The trained model would then help identify the suspiciousness of a given session in critical infrastructure. The data from adversary commands and session time analysis are, at best, probability-based. These should not be ideally marked as malicious by themselves. Therefore, qualitative criteria are chosen to provide avenues to appropriately rate the outputs from the machine learning models to determine the result.

The machine learning models described in this section are converted into Python language code snippets to aid data parsing from honeypots and critical infrastructure. This paper picks a threshold of 0.75 for the cosine similarity threshold to use a honeypot command for subsequent analysis. Please note that this threshold varies depending on factors such as the similarity between honeypots and critical infrastructure, the number of commands to match, and the risk sensitivity of an organization. Figure 5 represents the pseudocode to parse the data from the honeypots, and Figure 6 represents the pseudocode to parse the commands from the critical infrastructure and identify malicious events.

```

1  prompt = Custom_Prompt /*Custom Prompt from Section 3*/
2  vector_db = []
3
4  dedup_honeypot_logs = DEDUP(honeypot_logs)
5
6  FOR cmd IN dedup_honeypot_logs {
7      llm_response = FETCH_LLM_RESPONSE(PROMPT=prompt, INPUT_TEXT=cmd)
8      entities = FETCH_ENTITIES_IN_CMD(INPUT_TXT=cmd)
9      FOR entity in entities {
10          llm_response = REPLACE_ALL(TEXT=llm_response,
11                                      STRING_TO_REPLACE=entity,
12                                      REPLACED_STRING=entity.type.id)
13      }
14      cmd_vectors = VECTOR_PROCESS(MODEL=EMBEDDINGMODEL, llm_response)
15      vector_db.append({
16          cmd_val = cmd
17          cmd_description_vectors = cmd_vectors
18      })
19  }
20
21 SAVE(vector_db, PATH=VECTORDB_LOC)
22

```

**Figure 5.** Pseudocode to parse the honeypot data, leveraging LLM.

```

1  prompt = Custom_Prompt /*Custom Prompt from Section 3*/
2  query_prompt = Custom_Query_Prompt /*Custom Query Prompt from Section 3*/
3
4  vector_db = READ_FROM_SAVED_VECTORDB_FILE(PATH=VECTORDB_LOC)
5  user_analysis_db = [] // Store analysis DB results
6  dedup_user_logs = DEDUP(user_logs) // Dedup user logs to prevent running same cmd twice
7
8  FOR user_cmd IN dedup_user_logs {
9      llm_user_cmd_response = FETCH_LLM_RESPONSE(PROMPT=prompt, INPUT_TEXT=user_cmd)
10     entities = FETCH_ENTITIES_IN_CMD(INPUT_TXT=user_cmd)
11     FOR entity in entities {
12         llm_user_cmd_response = REPLACE_ALL(TEXT=llm_user_cmd_response,
13                                             STRING_TO_REPLACE=entity,
14                                             REPLACED_STRING=entity.type.id)
15     }
16     embedding_model_query = CREATE_MODEL_QUERY(llm_user_cmd_response)
17     user_cmd_vectors = VECTOR_PROCESS(MODEL=EMBEDDINGMODEL, embedding_model_query)
18     matches = COSINE_SIMILARITY(A=user_cmd_vectors, B=vector_db)
19     top_matches = TOP_K(matches, K=3) // Find top 3 matches
20     top_matches_exceeding_cossim_075 = EXTRACT_K(TOP_K=top_matches, COSSIM_THRESHOLD=0.75)
21     IF LEN(top_matches_exceeding_cossim_075 == 0) {
22         user_analysis_db.append({cmd = user_cmd, analysis = FALSE, key_category = NULL})
23     } ELSE {
24         llm_cmd_match_bool = FETCH_LLM_RESPONSE(PROMPT=query_prompt + top_matches_exceeding_cossim_075,
25                                               INPUT_TEXT=llm_user_cmd_response)
26         llm_cmd_category = FETCH_LLM_RESPONSE(PROMPT=query_prompt + top_matches_exceeding_cossim_075,
27                                               INPUT_TXT=llm_user_cmd_response)
28         user_analysis_db.append({cmd = user_cmd, analysis = llm_cmd_match_bool,
29                                 key_category = llm_cmd_category})
30     }
31 }
32
33 SAVE(user_analysis_db, PATH=USER_ANALYSIS_DB_LOC)
34

```

**Figure 6.** Pseudocode to parse the commands from critical infrastructure using LLM.

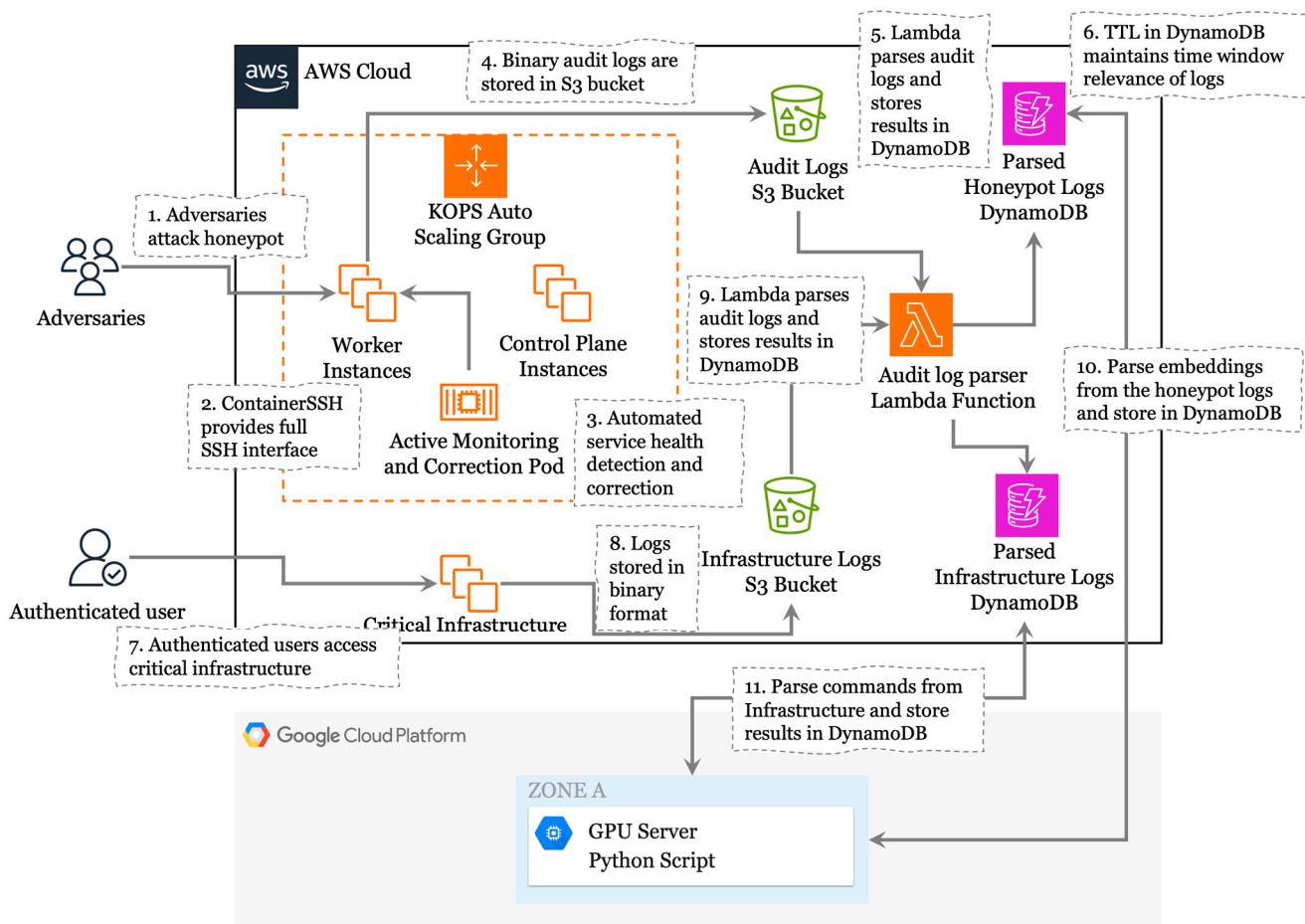
For the current setup and analysis of the events from the honeypot over two weeks, five tactics are shortlisted from MITRE ATT&CK categories: (1) Reconnaissance, (2) Persistence, (3) Impact, (4) Exfiltration, and (5) Command & Control.

### 3.6.2. Model Infrastructure Implementation

Figure 7 represents the high-level architecture for the current research. The following are the key design objectives identified for the honeypot and infrastructure design:

- System Similarity: The underlying honeypot and the critical infrastructure have similar setups. Both the underlying operating systems are Ubuntu 22.04 LTS Jammy Jellyfish. The software installed and the configuration settings are identical between both systems, preventing attackers from differentiating one system from another. Honeypot is configured to allow adversary login as any user and any password. Only one account credential used by critical infrastructure (account name: `ubuntu`) has a weak password (“`abc123`”);
- Attracting Adversaries: The AWS cloud hosts a combination of a honeypot network and critical infrastructure that authenticated and organization-approved users should use. Based on prior research, AWS was selected to host the honeypots because adversaries target cloud providers with the highest revenues compared to other cloud providers [19]. The key objective of the research is to maximize the attacks so that correlations to the honeypot data can happen at a shorter frequency;
- High-Interaction Setup: ContainerSSH (version 0.5) is the SSH proxy selected, allowing adversaries and system users to access the honeypot. ContainerSSH offers full SSH functionality to the end users [26] while acting as a proxy to record adversary and user commands. At the same time, the users receive a full-fledged shell into the host and can execute any command to their liking. The ContainerSSH proxy helps log all the SSH protocol-based interactions between adversaries and native infrastructure users;
- Availability Monitoring Detection and Correction: High-interaction honeypots expose the internals of the Operating System to adversaries. Due to the nature of the attacks executed on them, these honeypots are extremely fragile [27]. Therefore, additional infrastructure is needed to protect against service availability concerns. For this reason, Kubernetes service leveraging Kops (version 1.28.4) was selected to host the honeypot containers. Appropriate health checks are configured in Kubernetes to recycle the containers each time the health check results are degraded;
- Concurrent Log Storage: Logs from honeypots and critical infrastructure are quickly transmitted to an offsite S3 bucket (outside the VPC exposed to the users). Every session in both infrastructures is logged in the binary format through the ContainerSSH audit log. This binary format allows for the storage of all the SSH protocol communications and commands executed by the end user. The parity of the logging formats helps cross-correlate data between the two architectures. Further concurrent logging to offsite bucket helps ensure availability of the logs in the event of denial of service on the honeypot;
- Time Window Implementation: The Lambda service automatically parses all the logs from the S3 buckets and extracts the relevant information for the log analysis. The parsed honeypot data are stored in DynamoDB tables with an automated expiration TTL for each record. DynamoDB automatically deletes old records greater than an elapsed TTL value [28]. The current TTL selected for this research is 7 days. Hence, a current log record from critical infrastructure would be matched against honeypot logs from the past seven days;
- Machine Learning Analysis Compute: For the current research, the embedding model selected is “Salesforce/SFR-Embedding-Mistral”, and the LLM selected is “gpt-4-turbo”. The embedding model requires a heavy GPU for execution, and the code snippets need an Nvidia A100 GPU for smooth execution. Since the cost of constantly running the Nvidia A100 GPU  $24 \times 7$  is gigantic, a scheduler is used to launch the servers and code once every 24 h to parse the commands in the timeframe selected.

The servers then automatically shut down after the parsing for the day is complete. Since the parsing happens daily, it would take one day to identify any malicious activity in servers using this model. Please note that one can select smaller models instead of “Salesforce/SFR-Embedding-Mistral” for the analysis.



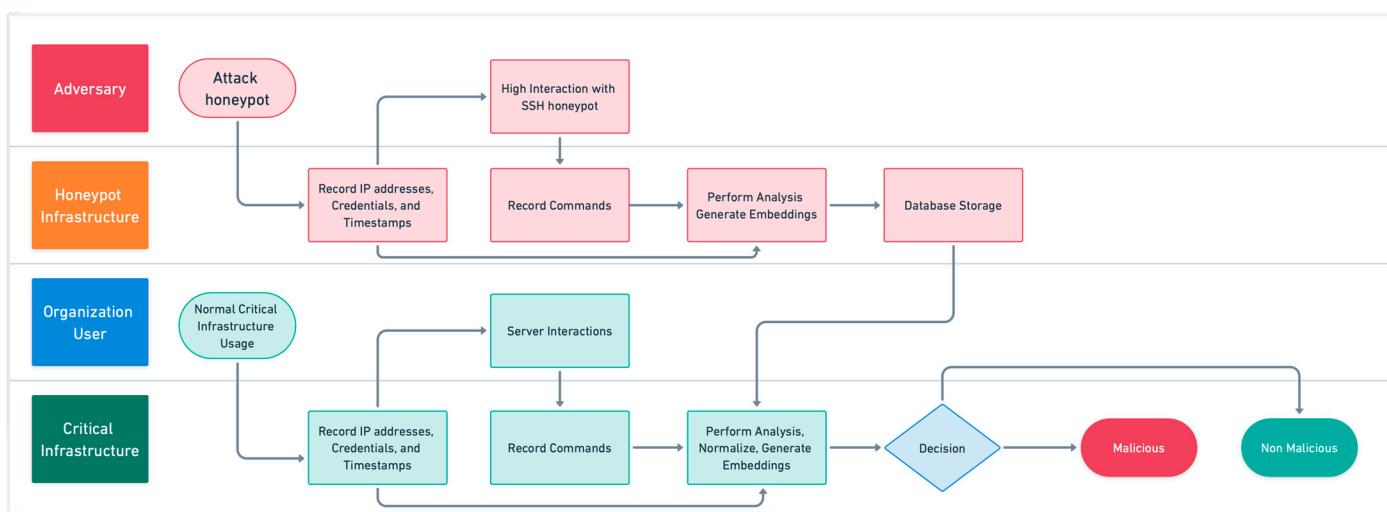
**Figure 7.** High-level architecture setup for the model validation.

### 3.6.3. Experimental Setup

- The experimental setup, depicted in Figure 7, was implemented in both AWS and GCP environments. Kops was employed to automatically create the necessary infrastructure within the AWS account. The Kubernetes API server was configured to be private and accessible only through a bastion host managed by Kops. Kubernetes was deployed on two instances: one dedicated to Kubernetes management and monitoring APIs and the other hosting containers for exploitation purposes. Monitoring capabilities within the Kubernetes service were enhanced using Grafana, Alert Manager, and Prometheus solutions. Alerts were configured to notify administrators if the CPU load on the EC2 instances exceeded 80% for a continuous period of 15 min.
- Kubernetes namespaces were created to host the SSH honeypot solution. The first namespace, named ContainerSSHAdmin, hosted the container SSH solution and its supporting infrastructure. The second namespace, named AdversaryTargets, hosted the target containers for SSH interactions. The containerized SSH service was created using Kubernetes deployment YAML files. Custom code was developed for the authentication service within ContainerSSH, allowing adversaries to access the high-interaction honeypot with any password or private key. The overall solution comprised two containers: one running ContainerSSH code acting as a proxy for SSH connections from adversaries and the other hosting the authentication service.

that accepted credentials submitted by the adversaries. The authentication service, invoked by ContainerSSH, created a container in the AdversaryTargets namespace for each successful authentication. Each successful SSH connection by an adversary instantiated a new container in the AdversaryTargets namespace, where the adversary obtained root privileges within the SSH session. To mimic critical infrastructure within the organization, Internet access from the SSH session was restricted.

- The ContainerSSH solution required access to S3 buckets, facilitated using an AWS access key and secret key pair securely stored within the ContainerSSH deployment. This setup ensured a robust and monitored environment for experimentation and analysis. Lambda functions were deployed on the AWS accounts to automatically parse logs arriving in the S3 buckets and write them to a new location within the same bucket. GCP servers equipped with A100 GPUs were deployed to analyze the logs and utilize them for LLM analysis. Scripts within Google Colab automatically downloaded parsed logs from the AWS S3 bucket and performed machine learning analysis leveraging the GPUs. Additionally, a Google Colab environment with A100 GPUs was employed for manual analysis and monitoring of the logs. Figure 8 depicts overall dataflow diagram of the setup.



**Figure 8.** Data flow diagram of the experimental setup.

- This comprehensive setup provided a secure and efficient infrastructure for conducting detailed experiments and analyses in a controlled environment. The honeypot and critical infrastructure were kept on the Internet without firewall controls for two weeks between 26 April 2024 and 3 May 2024. The availability of both systems during this time was observed to be 100%.

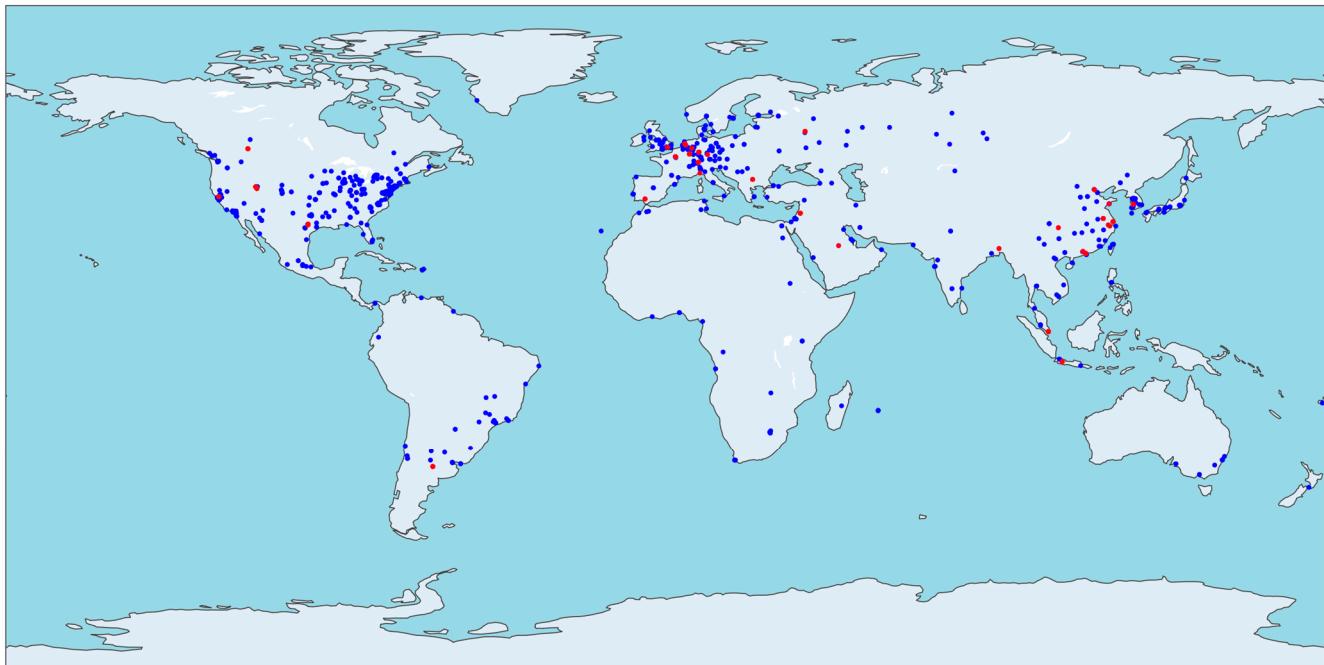
#### 4. Results and Discussion

The architected SSH honeypot and critical infrastructure servers remained open for attacks for 7 days, continuously collecting and analyzing data on the critical infrastructure while adversaries carried out attacks. The Kubernetes platform running the honeypot server helped stabilize it during the attacks.

##### 4.1. IP Address-Based Analytics

Attacks from over 1154 unique IP addresses were observed on the honeypot over 7 days. Figure 9 illustrates the geographic distribution of identified adversaries during this period. Among the 1154 IP addresses, only 345 unique IP addresses issued commands to the honeypot server (command executors). Other IP addresses scanned the service for usernames and passwords (scanners), presumably to launch attacks later once a database of vulnerable hosts is created. The blue dots in Figure 9 represent IP addresses primarily

functioning as scanners, while the red dots are IP addresses that attempted to exploit the honeypot with valid commands. The IP addresses seemed to be geographically distributed and thus cannot be statistically pinpointed to any location or geographic region. This is likely because attackers usually do not use their IP addresses to scan and maliciously connect to the exposed infrastructure. Instead, they leverage existing compromised hosts for such connections. Such hosts exist all over the world, and some of them are compromised using attacks similar to those observed on the honeypot. Any successful login from these IP addresses to the organization's critical infrastructure should be considered malicious and treated as an incident.



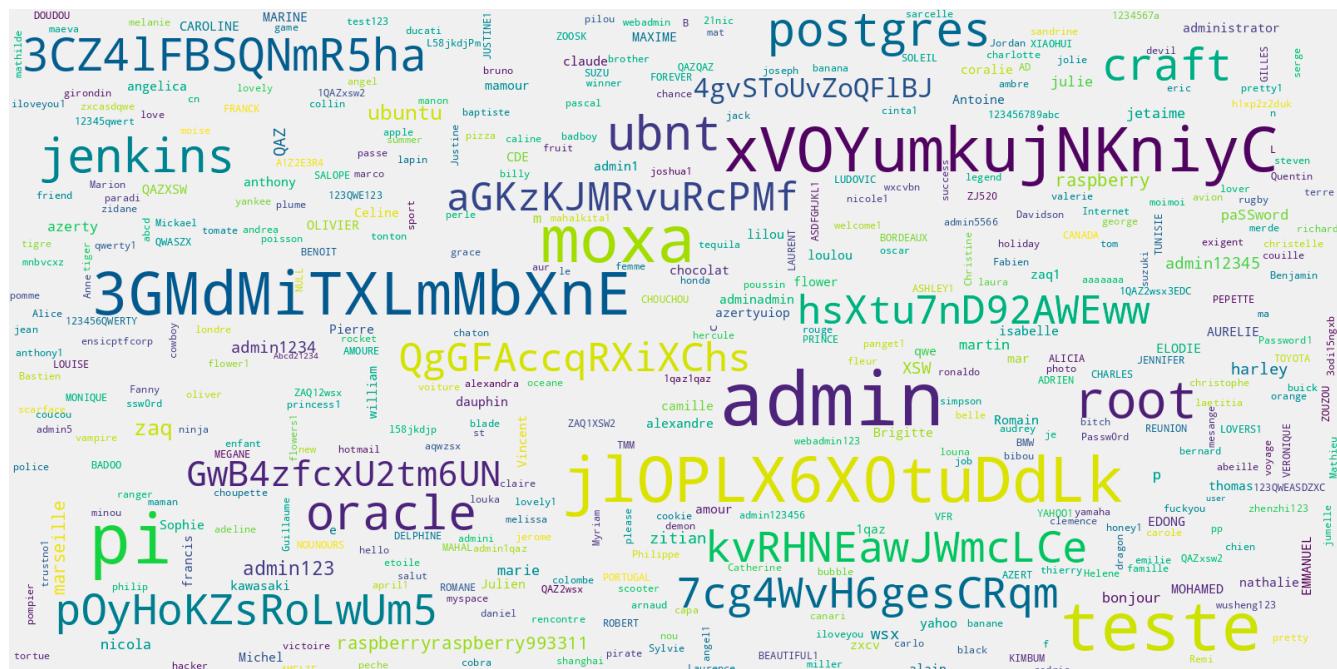
**Figure 9.** Geographic distribution of the attacks on the honeypot. Legend: • Recorded TCP connection without executing a single command in the honeypot (scanners). • Connection that executed commands in the honeypot (command executors).

#### 4.2. Credential-Based Analytics

Over the 7-day period, approximately 23,059 passwords were identified in the honeypot, with 10,274 unique strings recorded from unique IP addresses. Figure 10 displays a word cloud populated with the top 1000 unique strings from unique IP addresses and their observed frequencies within the honeypot. It is important to note that most passwords come from publicly available lists such as rockyou.txt [29]. Looking at the strings by holistic count rather than by strings used by unique IP addresses provides a different perspective. Figure 11 presents a word cloud representing the top 1000 passwords recorded in the honeypot based on their connection frequency from all IP addresses. This word cloud includes strings outside standard dictionary-based passwords, explicitly set by adversaries in previous connections to honeypots. Figure 12 depicts a sample attack in action, where an adversary from IP address 137.186.242.99 connected using the username and password "craft" and then attempted to modify the password to "GwB4zfcxU2tm6UN" in the honeypot. Subsequent connections from this IP address used the password "GwB4zfcxU2tm6UN" for connection to the honeypot. All passwords used by adversaries should be considered malicious, and if an adversary-used password is observed in the connection to critical infrastructure, the session must be considered malicious.



**Figure 10.** Word cloud of passwords from unique IP addresses. Size is proportional to number of logins.



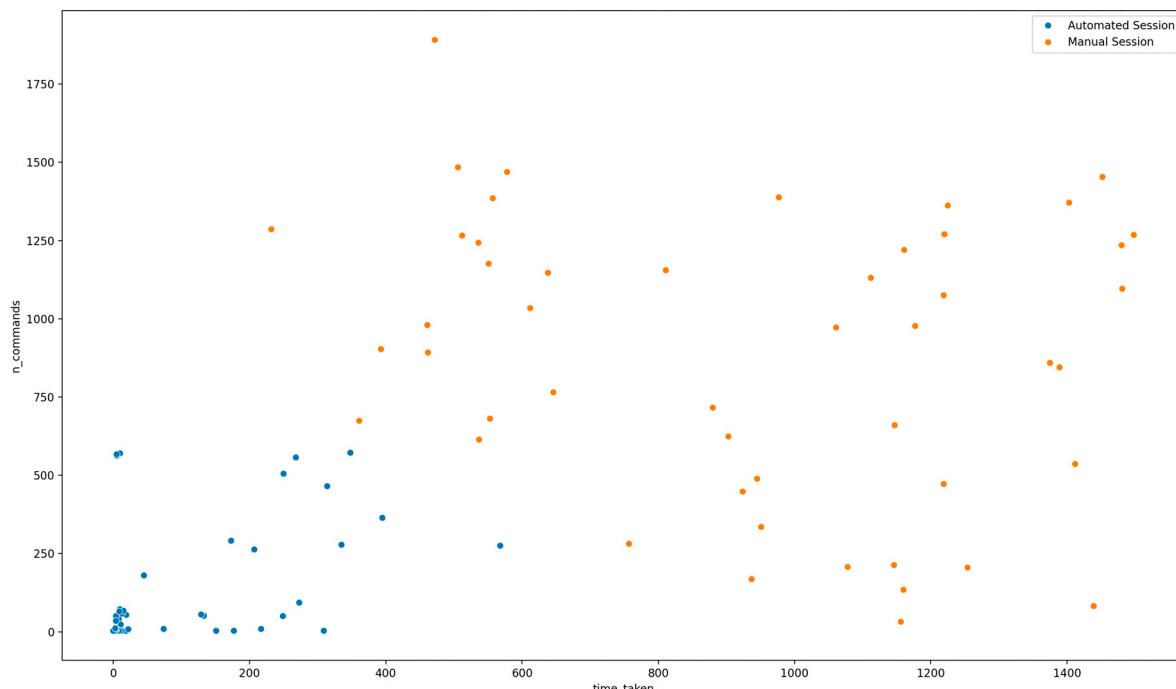
**Figure 11.** Word cloud of passwords. Size of the password in above picture is proportional to number of commands executed using the password.

```
{
    "src_ip": "137.186.242.99",
    "username": "craft",
    "password": "craft",
    "remote_entities": [
        "137.186.242.99"
    ],
    "command": "echo -e \\\"craft\\nGwB4zfcxU2tm6UN\\nGwB4zfcxU2tm6UN\\\" | passwd",
    "public_key": "",
    "latlng": [
        51.0501,
        -114.0853
    ]
},
{
    "src_ip": "137.186.242.99",
    "username": "craft",
    "password": "craft",
    "remote_entities": [
        "137.186.242.99"
    ],
    "command": "echo -e \\\"craft\\nGwB4zfcxU2tm6UN\\nGwB4zfcxU2tm6UN\\\" | passwd",
    "public_key": "",
    "latlng": [
        51.0501,
        -114.0853
    ]
}
}
```

**Figure 12.** Attack to change the password (used in word cloud in Figure 11) of user logged in subsequent session.

#### 4.3. Session Time-Based Analysis

Approximately 1902 sessions in the selected seven-day window had a session time greater than 0 s. During the same period, the critical infrastructure recorded around 51 SSH sessions. A K-Means plot generated two clusters from the data between the honeypot and the critical infrastructure. Figure 13 illustrates the K-Means plot, with data reevaluated for the observed time window. The blue dots represent the total time for automated sessions in the honeypot, and the orange dots represent the total time required for manual sessions in the critical infrastructure. The details of the K-Means plot are summarized in Table 7. The silhouette score of the K-Means clustering performed was calculated to be 0.98, indicating a very high clustering result with a good level of confidence in identifying an automated session versus a manual session in the cluster. Table 8 presents the confusion matrix of the K-Means plot observed from the clustering analysis, with an f1 score of 0.91.



**Figure 13.** Plot of clustering into two partitions using K-Means algorithm.

**Table 7.** Analysis of the K-Means plot.

Data Field	Honeypot Sessions (Automated)	Infrastructure Sessions (Manual)
Total number (n)	345	62
Maximum session time (seconds)	593	1498
Maximum commands in session (x)	552	1802
Minimum session time (seconds)	1	224
Minimum commands in session	1	80

**Table 8.** K-Means confusion matrix observed.

		Actual Values	
		Positive	Negative
Predicted Values	Positive	48	3
	Negative	7	1899

#### 4.4. Command Analysis

Adversaries executed 47,764 instances of commands over the time window on the honeypot machine. Bashlex analysis removed the variable entities within each command, revealing that many command instances were repeated with similar commands but with variable differences. Only 165 unique commands were identified among the 47,764 instances of commands executed in honeypots over one week. Table 9 details the top 10 commands executed in the honeypots. Notably, about 27,368 instances out of 47,774 instances of commands were a single echo command with specific arguments -e “\xF\x6B. This command typically prints “ok” on the screen; however, it is likely being used in attacker scripts to check if the shell environment responds to basic commands such as echo with an ASCII representation of hex characters. This finding correlates with research by Touch and Colin on commands executed within various honeypots on the Internet [30].

The 165 commands were analyzed using the “gpt-4-turbo” LLM with the prompt discussed in Section 3 to derive their behavior. Additionally, some post-processing was performed on LLM results using Bashlex functions to ensure that variables used in the command were entirely removed and replaced with generic strings. The resultant text was then converted to vectors using the “Salesforce/SFR-Embedding-Mistral” embedding model and stored in the DynamoDB database. The DynamoDB database has a TTL attribute set, ensuring the record automatically expires in a week.

Similar exercises were conducted for the commands executed in the critical infrastructure. Command details were derived from “gpt-4-turbo”, and descriptions derived from LLM were normalized with variables removed using a Python script leveraging the Bashlex library. The resultant text was then converted to vectors using the previously mentioned embedding model—“Salesforce/SFR-Embedding-Mistral”. A cosine similarity search was performed on the vectors obtained from commands in the critical infrastructure against the list of vectors from the honeypot executed by the adversaries. Table 10 provides examples of the commands and their cosine similarity search results.

**Table 9.** Commands executed on the honeypot over one week.

Variable Normalized Command Template	Example Command	# Repeats
echo -e STRING	echo -e "\x6F\x6B"	27,368
rm -rf .bash_history	rm -rf .bash_history	4080
kill -9 \$(ps aux   grep STRING   grep -v grep   awk '{print \$2}')	kill -9 \$(ps aux   grep xrx   grep -v grep   awk '{print \$2}')	2832
rm -rf STRING	rm -rf .bash_history	3060
touch STRING	touch secure	1700
unset STRING	unset HISTFILE	1020
uname -a	uname -a	381
history -n	history -n	340
nohup sh STRING &	nohup sh/tmp/.ssh/b &	340
cat > STRING	cat > systemd-net	318
echo -e STRING	echo -e "\x6F\x6B"	27,368
rm -rf .bash_history	rm -rf .bash_history	4080

**Table 10.** Cosine similarity analysis results.

Commands from Critical Infrastructure	Top 3 Matched Commands	Top 3 Cosine Similarity Scores
tac/etc/passwd   grep ubuntu	1. cat/etc/passwd   grep ubuntu 2. grep ubuntu/etc/passwd 3. ul/etc/passwd   grep ubuntu	0.9080 0.8951 0.8661
pip install requests	1. apt install sudo curl -y 2. python -c "print(open('/etc/passwd','r').read())"   grep ubuntu 3. curl https://ipinfo.io/org --insecure -s	0.7136 0.6556 0.6404
id ubuntu	1. id -u ubuntu 2. getent passwd ubuntu 3. cat/etc/passwd   grep ubuntu	0.8081 0.7995 0.7720
ps aux	1. ps   grep '[Mm]iner' 2. ps -ef   grep '[Mm]iner' 3. uname -a	0.7525 0.7402 0.7236
curl https://virustotal.com/api	1. curl https://ipinfo.io/org --insecure -s 2. CATCMD = cat; eval "\$CATCMD/etc/passwd 3. ./systemd-net --opencl --cuda -o 142.202.242.45:80 -u 43UdT-A- JHLU2jN3knid9Vq1GB625ZCKhZiZVDZ8Mi PUWQYXbq9QmmAq79BoFhqdLhoygCLsBue6gpoXB kSw5GVW7ZGrXyauv -p xxx -k --tls --tls-fingerprint 420c7850e09b7c0bdcf748a7da9eb3647daf8515718f36d 9ccfdd6b9ff834b14 --donate-level 1 --background	0.7507 0.6875 0.6776

Commands with a cosine similarity score > 0.75 were then sent to “gpt-4-turbo” to validate whether the commands in the critical infrastructure matched the commands with a high cosine similarity score and to fetch the correct category for the attacker TTP. Table 11 presents the LLM model “gpt-4-turbo” results for the examples selected in Table 10. Due to

the use of LLM at this step, even though the cosine similarity scores exceeded the threshold, sometimes the command was not malicious, or the malicious category was not identified. In these circumstances, running LLM helped refine the results from a similarity search.

**Table 11.** Illustrative LLM model results to categorize commands and fetch TTP Categories.

Commands from Critical Infrastructure	Top Matched Commands with Cosine Similarity Score > 0.75	LLM—Is It Similar to a Malicious One?	LLM—Identify TTP Category
tac/etc/passwd   grep ubuntu	1. cat/etc/passwd   grep ubuntu 2. grep ubuntu/etc/passwd 3. ul/etc/passwd   grep ubuntu	TRUE	Information Gathering
pip install requests	NA	NA (Threshold cosine similarity not met)	NA (Threshold cosine similarity not met)
id ubuntu	1. id -u ubuntu 2. getent passwd ubuntu 3. cat/etc/passwd   grep ubuntu	TRUE	Information Gathering
ps aux	1. ps   grep '[Mm]iner'	TRUE	Information Gathering
curl https://virustotal.com/api	1. curl https://ipinfo.io/org --insecure -s	FALSE	None

#### 4.5. Identifying Malicious Command Execution in Critical Infrastructure

Combining the above criteria, a list of criteria to identify malicious command execution in critical infrastructure is as follows:

1. A successful connection and login to critical infrastructure from a known attacker's IP address recorded in the honeypot during the observation time window could be deemed malicious;
2. A successful login into critical infrastructure leveraging credentials used by adversaries in the honeypot during the observation window could be deemed malicious;
3. Table 12 presents the qualitative weights for identifying malicious command execution between session time and command analysis.

**Table 12.** Qualitative criteria between machine learning models.

LLM Command Match	LLM Category	K-Means Analysis	Final Decision
TRUE	Reconnaissance	TRUE	Malicious
TRUE	Persistence	TRUE	Malicious
TRUE	Impact	TRUE	Malicious
TRUE	Exfiltration	TRUE	Malicious
TRUE	Command and Control	TRUE	Malicious
TRUE	Reconnaissance	FALSE	Benign
TRUE	Persistence	FALSE	Benign
TRUE	Impact	FALSE	Benign
TRUE	Exfiltration	FALSE	Malicious
TRUE	Command and Control	FALSE	Malicious
FALSE	ANY	TRUE	Benign
FALSE	ANY	FALSE	Benign

By following these algorithms, one can quickly parse high-interaction honeypot logs on an ongoing basis to establish if a given event in an organization's critical infrastructure is malicious.

## 5. Remarks and Conclusions

A robust model has been designed to quickly parse the high-interaction honeypot logs and associate them with activities within the organization's infrastructure automatically, leveraging machine learning and large language model-based techniques. The model was tested using real-world attacks on a high-interaction honeypot, and the commands were compared to those executed on real-world SSH servers running in the AWS cloud. The attackers' motives to quickly exploit and establish persistence in Internet-exposed servers drive their attack tactics and the similarity between various servers that expose the same protocol. It is possible to draw the following conclusions by leveraging the research in this article.

Four independent parameters were used to analyze adversary behavior in high-interaction honeypots. These four parameters would help shape the monitoring implemented in the critical infrastructure:

1. Two of the parameters, the adversary's IP address and credentials, are direct indicators of compromise. Essentially, if the connections to the critical infrastructure originate from previously recorded adversary IP addresses or leverage one of the attacker's compromised credentials, the activity in the critical infrastructure can be deemed malicious;
2. K-Means clustering has proven to be an effective method for analyzing session time on honeypot servers and critical infrastructure. Attackers spend minimal time executing commands to gather information and establish persistence in the honeypot [19]. K-Means clustering helps derive clusters that identify honeypot sessions against typical server sessions with an excellent silhouette score and F1 score;
3. The Retrieval and Augmented Generation (RAG) method offers a mechanism for identifying if commands executed in critical infrastructure are malicious, leveraging the information from existing commands executed in a honeypot infrastructure. Further, the LLM processing also classifies a command executed in the critical infrastructure into various MITRE ATT&CK categories;
4. A qualitative combination of K-Means analysis and RAG analysis using language model-based techniques helps derive another avenue for categorizing events in critical infrastructure as malicious.

### Further Research and Next Steps

Previously, data analysis from high-interaction honeypots could not be automated. By leveraging the research in this paper, any organization can process the data from high-interaction honeypots and detect incidents in their infrastructure as early as possible. Faster detection aids in reducing incident costs. This research would help organizations make automated decisions and stay aware of environmental threat actors.

Once an attack is deemed malicious, organizations can define plans to enhance their incident management efforts to contain it. Further research is needed to develop a robust incident management plan that aligns with the discussions in this paper to aid in analyzing, containing, mitigating the threat, and remediating any underlying security issues. The current research aims to understand adversary TTPs leveraging attacks on the application layer in the networking layer. Further research is required to extend the methodology and design proposed in the current research to identify DoS attacks executed on non-application layers (e.g., ping flood, TCP SYN attacks) in the networking stack. The research could also be extended to include a more complex honeypot setup involving multiple honeypot servers to allow for lateral movement of adversaries and to leverage honeytokens to add new dimensions to the analysis. Normalizing this research across multiple different

honeypot scenarios over an extended time is necessary to capture additional analysis avenues and enhance security threat research for organizations.

**Author Contributions:** Conceptualization, P.L., K.G. and C.V.; methodology, P.L.; software, P.L. and K.G.; validation, C.V. and K.G.; formal analysis, P.L.; investigation, P.L.; resources, C.V. and K.G.; data curation, P.L.; writing—original draft preparation, P.L.; writing—review and editing, P.L.; visualization, P.L.; supervision, C.V.; project administration, P.L. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** The datasets generated and/or analyzed in the current study are available from the corresponding authors upon reasonable request.

**Acknowledgments:** We acknowledge OpenAPI and GCP cloud platforms for their support in this project. We also thank the model authors for Salesforce/SFR-Embedding-Mistral for their work on the embedding model used in the research.

**Conflicts of Interest:** The authors declare that they have no known competing financial interests or personal interests that could have appeared to influence the work reported in this paper.

## References

1. Rising Cyber Threats Pose Serious Concerns for Financial Stability. Available online: <https://www.imf.org/en/Blogs/Articles/2024/04/09/rising-cyber-threats-pose-serious-concerns-for-financial-stability> (accessed on 24 April 2024).
2. Data Breach Action Guide. Available online: <https://www.ibm.com/reports/data-breach-action-guide> (accessed on 24 April 2024).
3. COVID-19 Continues to Create a Larger Surface Area for Cyberattacks. Available online: <https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/docs/vmwcb-report-covid-19-continues-to-create-a-larger-surface-area-for-cyberattacks.pdf> (accessed on 24 April 2024).
4. Impact of COVID-19 on Cybersecurity. Available online: <https://www2.deloitte.com/ch/en/pages/risk/articles/impact-covid-cybersecurity.html> (accessed on 24 April 2024).
5. What's the Difference Between a High Interaction Honeypot and a Low Interaction Honeypot? Available online: <https://www.akamai.com/blog/security/high-interaction-honeypot-versus-low-interaction-honeypot-comparison> (accessed on 24 April 2024).
6. High Interaction Honeypot. Available online: <https://www.sciencedirect.com/topics/computer-science/high-interaction-honeypot> (accessed on 24 April 2024).
7. Ilg, N.; Duplys, P.; Sisejkovic, D.; Menth, M. Survey of Contemporary Open-Source Honeypots, frameworks, and tools. *J. Netw. Comput. Appl.* **2023**, *220*, 103737. [[CrossRef](#)]
8. 2023 Honeypotting in the Cloud Report: Attackers Discover and Weaponize Exposed Cloud Assets and Secrets in Minutes. Available online: <https://orca.security/resources/blog/2023-honeypotting-in-the-cloud-report/> (accessed on 24 April 2024).
9. Hacking With GPT-4: Generating Obfuscated Bash Commands. Available online: <https://www.linkedin.com/pulse/hacking-gpt-4-generating-obfuscated-bash-commands-jonathan-todd/> (accessed on 24 April 2024).
10. Generative AI to Become a \$1.3 Trillion Market by 2032, Research Finds. Available online: <https://www.bloomberg.com/company/press/generative-ai-to-become-a-1-3-trillion-market-by-2032-research-finds/> (accessed on 24 April 2024).
11. Liu, Y.; Cao, J.; Liu, C.; Ding, K.; Jin, L. Datasets for Large Language Models: A Comprehensive Survey. *arXiv* **2024**, arXiv:2402.18041. [[CrossRef](#)]
12. Top Threats You Need to Know to Defend Your Cloud Environment. Available online: <https://www.crowdstrike.com/blog/adversaries-increasingly-target-cloud-environments/> (accessed on 24 April 2024).
13. No, G.; Lee, Y.; Kang, H.; Kang, P. RAPID: Training-free Retrieval-based Log Anomaly Detection with PLM considering Token-level information. *arXiv* **2023**, arXiv:2311.05160. [[CrossRef](#)]
14. Karlsen, E.; Luo, X.; Zincir-Heywood, N.; Heywood, M. Benchmarking Large Language Models for Log Analysis, Security, and Interpretation. *arXiv* **2023**, arXiv:2311.14519. [[CrossRef](#)]
15. Guu, K.; Lee, K.; Tung, Z.; Pasupat, P.; Chang, M. Retrieval Augmented Language Model Pre-training. In Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, 13–18 July 2020. [[CrossRef](#)]
16. Yang, X.; Yuan, J.; Yang, H.; Kong, Y.; Zhang, H.; Zhao, J. A Highly Interactive Honeypot-Based Approach to Network Threat Management. *Future Internet* **2023**, *15*, 127. [[CrossRef](#)]
17. Szabó, Z.; Bilicki, V. A New Approach to Web Application Security: Utilizing GPT Language Models for Source Code Inspection. *Future Internet* **2023**, *15*, 326. [[CrossRef](#)]
18. Wang, B.-X.; Chen, J.-L.; Yu, C.-L. An AI-Powered Network Threat Detection System. *IEEE Access* **2022**, *10*, 54029–54037. [[CrossRef](#)]

19. Lanka, P.; Varol, C.; Burns, K.; Shashidhar, N. Magnets to Adversaries—An Analysis of the Attacks on Public Cloud Servers. *Electronics* **2023**, *12*, 4493. [[CrossRef](#)]
20. Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.T.; Rocktäschel, T.; et al. Retrieval-augmented Generation for Knowledge-intensive NLP Tasks. *Adv. Neural Inf. Process Sys.* **2020**, *33*, 9459–9474.
21. Amatriain, X. Prompt design and engineering: Introduction and Advanced Methods. *arXiv* **2024**, arXiv:2401.14423. [[CrossRef](#)]
22. Bashlex—Python Parser for Bash. Available online: <https://github.com/idank/bashlex> (accessed on 24 April 2024).
23. Overall MTEB English Leaderboard. Available online: <https://huggingface.co/spaces/mteb/leaderboard> (accessed on 24 April 2024).
24. SFR-Embedding-Mistral: Enhance Text Retrieval with Transfer Learning. Available online: <https://blog.salesforceair.research.com/sfr-embedded-mistral/> (accessed on 24 April 2024).
25. Enterprise Matrix. Available online: <https://attack.mitre.org/versions/v15/matrices/enterprise/> (accessed on 24 April 2024).
26. ContainerSSH: Launch Containers on Demand. Available online: <https://containerssh.io/v0.5/getting-started/> (accessed on 24 April 2024).
27. Jiang, X.; Wang, X. “Out-of-the-Box” Monitoring of VM-Based High-Interaction Honeybots. *Adv. Neural Inf. Process Syst.* **2007**, *4637*, 198–218. [[CrossRef](#)] [[PubMed](#)]
28. Amazon DynamoDB Developer Guide—Time to Live (TTL). Available online: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/TTL.html> (accessed on 24 April 2024).
29. One Of The 32 Million With A RockYou Account? You May Want To Change All Your Passwords. Like Now. Available online: <https://techcrunch.com/2009/12/14/rockyou-hacked/> (accessed on 24 April 2024).
30. Touch, S.; Colin, J.-N. A Comparison of an Adaptive Self-Guarded Honeypot with Conventional Honeybots. *Appl. Sci.* **2022**, *12*, 5224. [[CrossRef](#)]

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.