2025 IEEE 14th International Conference on Communication Systems and Network Technologies (CSNT) | 979-8-3315-3193-5/25/$31.00 ©2025 IEEE | DOI: 10.1109/CSNT64827.2025.10968591

Generative AI SSH Honeypot With Reinforcement Learning

Pragna Prasad, Nikhil Girish, Sarasvathi V, Apoorva VH, Prem Kumar S
Department of CSE
PES University
Bengaluru, India
pragpras24@gmail.com, nikhilgirish22@googlemail.com, sarsvathi@pes.edu, vhapoorva@gmail.com, premkumarsdps@gmail.com

***Abstract*—In today's rapidly evolving cyber landscape, traditional honeypots face significant limitations in their ability to adapt to sophisticated attack patterns and maintain convincing interactions with attackers. We propose Generative AI SSH Honeypot (GASH): an innovative approach to this problem combining Deep Reinforcement Learning (DRL) with Large Language Models (LLMs) to create a dynamic, engaging honeypot system. GASH employs a Deep Q-Network (DQN) architecture for strategic decision-making and leverages OpenAI's GPT-4o for generating contextually appropriate responses while maintaining robust security measures. Our experimental results demonstrate improvements in attacker engagement duration compared to traditional honeypot systems like Kippo and Cowrie.**

Keywords- honeypot, reinforcement learning, large language models, cybersecurity, SSH, AI

I. INTRODUCTION

Honeypots are a defensive security mechanism designed primarily to deceive attackers into engaging with them in order to fulfil two main objectives; to protect real systems from malicious actions by attackers, and simultaneously gain information about the techniques and tools utilized by attackers. The earliest mention of a similar concept is in the Federal Information Processing Standards 39 or FIPS 39 from 1976 [1]. Early implementations of honeypots were simple decoy systems providing basic emulation of network services. However, as cyber threats have become more sophisticated, with attackers employing automated tools and artificial intelligence, the limitations of traditional honeypots, which rely primarily on static responses and predetermined interaction patterns, become more apparent. Traditional static security approaches have proven inadequate against sophisticated modern threats, necessitating more dynamic and adaptive defence mechanisms [2].

The financial impact of cyberattacks continues to grow, with a predicted estimate of global cybercrime damage exceeding $10.5 trillion annually by 2025, up from $3 trillion in 2015 according to the 2023 Official Cybercrime Report [3]. An escalating threat landscape like this necessitates more sophisticated defensive tools. While traditional honeypots still remain useful, they are easy to identify for experienced attackers, and grow unable to cope with the complexity of modern threats – limiting their usefulness as research tools as well as security measures. Early honeypot implementations like Honeyd [4] provided virtual honeypot services that could simulate multiple virtual hosts on a network simultaneously, making them effective for large-scale deployment. The emergence and adoption of machine learning and artificial intelligence in the field of cybersecurity opens up new pathways. Recent advances in reinforcement learning and natural language processing, namely in the form of Large Language Models (LLMs), present opportunities to create more dynamic, engaging, and adaptive honeypot systems. These technologies can enable development of systems capable of learning from their interactions with attackers and thus, adapting their behaviour to better engage them, as well as generating more convincing responses to attacker actions, all while maintaining the security and integrity of the system it is running on.

A. Current Challenges

The static rule-based responses that are responsible for the limited adaptability of traditional honeypots to new attack patterns and attacker tactics, present a significant challenge to their utility, as they leave these systems unable to adapt to novel and sophisticated attack techniques. The lack of automated update mechanisms for new and emerging threat types further compounds this issue, leaving honeypots, especially high interaction ones that utilize real systems, vulnerable.

Predictable response behaviours constitute another major limitation. These predetermined response patterns are easily recognizable by attackers. Utilization of automated attack tools capable of detecting honeypots further exacerbates the problem of interactivity and successful deception. Often, systems struggle with holding contextually appropriate conversations, and rigid command execution paths they offer typically lack the authenticity needed to convince sophisticated attackers and the tools they use. The challenge of insufficient engagement with such sophisticated attackers, manifests in the major way of limited interactions; attackers curtail their interaction with the honeypot, reducing their efficacy and limiting the information gained about attackers, and their tactics and tools. The inability to simulate complex system behaviours, without the risk of using a real system that can be compromised and present an attack vector itself,

979-8-3315-3193-5/25/$31.00 ©2025 IEEE
DOI: 10.1109/CSNT.2025.129

coupled with the lack of realistic error handling and responses, makes it difficult to maintain the illusion of a genuine system.

Any limitation in a honeypot's capability to gather detailed threat intelligence represents a significant operational weakness for the same. The basic logging mechanisms these systems typically employ often forgo crucial attack. Insufficient metadata collection for threat analysis, reduces the value of gathered intelligence.

High maintenance requirements of high-interaction honeypots further pose ongoing operational challenges. Such systems require regular manual updates in order to maintain believability and security. This resource-intensive configuration management processes, when coupled with the attached difficulty in scaling across multiple instances, can make these systems challenging to maintain effectively-especially for smaller organizations limited in the resources available to them.

B. *Contribution*

This paper presents several significant contributions to the field:

- A novel architecture that combines Deep Reinforcement Learning with LLMs for honeypot systems
- An improved decision-making mechanism to handle attacker input using Deep Q-Network
- Integration of GPT-4o turbo for dynamic response generation to increase interactivity
- A comprehensive logging and analysis system

II. LITERATURE SURVEY

Honeypots have been widely used in cybersecurity to study attacker behaviour, gather intelligence, and protect real systems by luring malicious actors into interacting with fake environments. Early honeypots like Kippo [5] and Cowrie [6] laid the foundation by emulating low- and medium-interaction SSH environments. While effective in capturing basic data, these systems were static in their responses and lacked the adaptability required to counter evolving attack strategies. The introduction of adaptive honeypots addressed these limitations. Tools like LaBrea [7] and THP (Tiny Honeypot) [8] were among the earliest attempts at creating deceptive network services, focusing primarily on TCP/IP level interactions.

Heliza [9], one of the first systems to incorporate reinforcement learning (RL), utilized SARSA and a Markov state model to dynamically adjust its responses based on attacker actions. While Heliza provided novel adaptability, it was constrained by its computational complexity and inability to scale effectively.

RASSH [10] demonstrated improved adaptability compared to static systems. It employed PyBrain for implementing SARSA algorithms, integrating a modular architecture that included a honeypot module, an actions module, and an RL module. Despite its innovative design, RASSH faced limitations in scalability and functionality due to reliance on older libraries and techniques. QRASSH extended RASSH by incorporating Deep Q-Learning, enabling more efficient learning from attack patterns and improving the realism of interactions. By emulating a full-fledged SSH server and leveraging neural networks, QRASSH enhanced attacker engagement and data collection, though its computational demands increased significantly [11].

Research work by Doubleday et al. [12] provided comprehensive guidelines for building and analysing SSH honeypots, establishing foundational practices for honeypot deployment and maintenance.

The shift toward high-interaction honeypots introduced systems like AIIPot and IoTCMal, which targeted IoT environments. AIIPot combined RL and transformer models to simulate realistic interactions and adapt dynamically to attacker behaviour, thereby capturing detailed attack patterns [13]. IoTCMal, a hybrid honeypot, combined low- and high-interaction components to analyse malware targeting IoT devices [14]. These systems succeeded in improving engagement and data collection but faced challenges such as resource overhead and potential model poisoning.

Recent work by Boffa et al. [15] has explored the potential of natural language processing for analysing honeypot logs, enabling better understanding of attacker behaviours and patterns. Docker-based containerization [16] has emerged as a promising approach for creating isolated, high-interaction honeypot environments, as demonstrated by Buzzio-Garcia's work.

The integration of large language models (LLMs) into honeypots marked a significant advancement in realism. Systems like LLM-based honeypots used pre-trained models to dynamically generate human-like responses, thereby increasing attacker trust and engagement [17]. However, these systems struggled with adaptability, as their interaction quality depended heavily on static language model training. Additionally, latency and stochasticity in LLM responses sometimes hindered real-time interaction.

Combining RL and LLMs has shown promise in bridging these gaps. For instance, efforts to integrate semantic embeddings with RL models demonstrated improved engagement and adaptive learning [18]. However, existing systems often fail to balance high interaction with adaptability, leaving opportunities for improvement. Comparative studies between adaptive and conventional honeypots [19] have shown significant improvements in attacker engagement and deception capabilities.

The proposed GASH (Generative AI SSH Honeypot) system builds on this prior research by integrating RL for dynamic decision-making and LLMs (via the OpenAI API) for realistic, context-aware interaction. Unlike static or semi-adaptive systems, GASH is designed to maximize attacker engagement while providing actionable intelligence. By leveraging advancements from RASSH, QRASSH, and LLM-based systems, GASH aims to address the dual challenges of adaptability and interaction quality, offering a scalable and efficient solution for modern cybersecurity threats.

III. PROPOSED METHODOLOGY

The structure of GASH (Generative AI SSH Honeypot) is proposed to be one that strikes a balance between decision-making process using the RL model, and engaging and dynamic response generation utilising the OpenAI API. The use of reinforcement learning in GASH draws inspiration from evaluations conducted by Kristyanto et al., [20] who explored RL algorithms for optimizing SSH honeypots. The architecture diagram is illustrated in Figure 1.

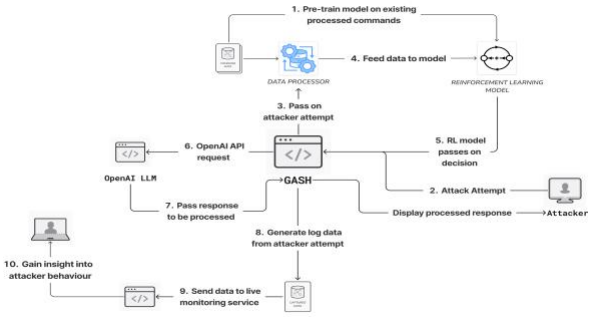

Figure 1. GASH Architecture Diagram

The flow of the project is as follows: malicious actors attempt to attack the honeypot system. All of these incoming commands entered in the honeypot are captured and cleaned for further use. Some basic error handling mechanisms are also introduced here and these act upon the input. The commands are then pre-processed by breaking down the input into smaller units for greater efficiency in handling them. After they are so tokenized, they are sent to the RL model or the decision-making engine.

This engine utilises a Deep Q-Network (DQN) architecture to measure the current state and select the most rewarding action. The RL model takes in the cleaned and pre-processed attacker input and arrives at the most optimal decision to be taken based on its learnings from the training session that takes place before the SSH server is live. In the training itself, rewards are given for keeping the attacker engaged for longer periods of time. In parallel, rewards are also given when the model blocks certain harmful commands, allows harmless commands, gives a delay for commands that, in a real system, would warrant a few seconds before the response is displayed and so on.

Algorithm 1: DQN Agent Training and Action Selection

Input: State, action, reward, next state, done flag
Output: Optimal action, updated model

1. **Initialize** DQN model and target model with *stateSize* and *actionSize*.
2. **Initialize** parameters: *learningRate, epsilon, gamma, memoryBuffer, and optimizer*.
3. Update Target Network: Copy weights from the DQN model to the target model.
4. **for** each *step*:
   Action Selection:
      if *randomValue* < *epsilon*:
         Select a random *action*.
      else:
         Select the *action* with the highest Q-value from the model.
5. **Store** the experience (*state, action, reward, nextState, done*) in memory.
6. **if** memory contains enough *experiences* (≥ *batchSize*):
   Sample *minibatch* of *experiences* from memory.
   for each *experience*:
      Compute the target Q-value:
      if *done*: *target* = *reward*.
      else: *target* = *reward* + *gamma* * *max* (next state Q-value from target model).
   Optimize model by minimizing the *loss* between current Q-value and target Q-value.
7. Epsilon Decay: **Decrease** *epsilon* value (exploration) after each episode until *epsilon_min* is reached.
8. **Save** Model: **Save** *model weights* to file.
9. **Load** Model: **Load** *model weights* from file and update target model

Algorithm 2: Reinforcement Learning Trainer

Input: Environment, OpenAI API, DQN Agent
Output: Trained agent, performance analysis

1. **Initialize** trainer by setting up the honeypot environment using a predefined list of commands.
2. **Create** a command handler leveraging the OpenAI API for generating dynamic responses.
3. **Initialize** the *DQNAgent* for decision-making.
4. **Agent Training**: for each episode:
   Reset the environment to start fresh and retrieve the initial state.
   Initialize *totalReward* to track cumulative reward for the episode.
   for each *step* within the episode:
      Use the agent to select an action based on its policy.
      Execute the action in the environment and observe the outcomes:
      nextState: Resulting state after the action. reward: Feedback signal for the action. *done*: Whether the session has terminated.
      Train the agent using the observed experience: (state, action, reward, *nextState*, done).
      Update the current state to *nextState*.
      Add the reward to *totalReward*.
      if *done* = True: terminate the episode.
   Record *totalReward* for the episode and display progress.
   Save the trained model after all episodes are completed. Plot the rewards accumulated across episodes.
3. **Model Persistence**: Save the agent's learned model weights to a file for reuse.
   Load previous model weights from a file to initialize the agent.

4. **Training Performance Analysis**: Plot the total rewards for each episode to visualize training progress.

Compute and plot a moving average of the rewards for a smoother trend.

Analyse and display the frequency of each action taken during training.

One out of five actions can be taken by the RL model for a given command: **allow, block, delay, fake** and **insult**. Fake is the decision taken when some deception needs to be introduced and insult acts as a Reverse Turing Test, to differentiate actual human attackers from bots. Penalties are given when the model allows harmful commands, when it makes the attacker leave the session, when they realize that this is a honeypot etc. The resulting decision, combined with the attacker input, context, personality file and any static response, is then sent to the OpenAI API.

TABLE I. REWARD STRUCTURES FOR COMMAND GROUPS IN THE HONEYPOT ENVIRONMENT

Command Category	***Example Commands***	***Description***	***Risk Assessment***	***Reward Structure***
File System Operations	ls, cd, mkdir, rmdir, touch	File management and system info commands	Low	Allow (+20) / Block (-20)
System Administration	chmod, chown, passwd, useradd, usermod, su	Manage system permissions and settings	Medium	Block (+15) / Fake (-25)
System Control	shutdown, reboot, kill, killall, pkill, crontab, at	Potentially damaging commands to the system	High	Block (+15) / Insult (-30)
Network Operations	ping, curl, wget, scp, sftp	Related to network interaction and surveillance	Medium	Delay (+15) / Fake (-20)
Resource monitoring	uptime, vmstat, ps, iostat	Provide system performance and resource metrics	Low	Fake (+10) / Allow (-15)
Custom commands	./exploit, ssh-keygen, ssh-agent, sshd, ssh-copy-id, ssh-add	Scripts or tools used for exploitation or advanced interaction	High	Insult (+1) / Allow (-47)

TABLE II. COMPARISON OF NUMBER OF COMMANDS TRAINED ON

Honeypot	***Number of commands***	***Description***
QRASSH	57	Includes basic Linux commands and exit. Focuses on minimal interaction to analyse attacker behaviour
RASSH	75	Extends command support, including common Linux commands and specific system management commands like *useradd* and *userdel*
GASH	101	Supports a custom set of Linux commands categorized into file system, system commands, package management, and network-related commands, optimized for prolonging session durations

We aim to enhance the honeypot's realism by generating dynamic and engaging context-aware responses via the OpenAI API. Here, we utilize OpenAI's GPT-4o model to produce extremely personalized responses that convinces the attacker and any tools they may be using that the GASH system is a real-world system. This also helps in increasing the period of time attacker remain on the system, which in turn helps network administrators analyse common attacker behaviours. In the prompt that is sent to the API, we wrap up the command entered by the attacker, the decision taken for the same by the RL model, any static responses to use as template, context of the system, a personality file that tells the LLM exactly what it is supposed to do to prevent model drift/hallucination. The response generated is then displayed to the attacker.

The attack interface handles attacker inputs, displays outputs from the API, and generally imitates a legitimate system to look as authentic as possible. This also captures and builds comprehensive logs.

Finally, we analyse log data from the interactions of the honeypot system. We record each command that is executed, the timestamp when it was executed, the source IP address, the decision taken by the RL model and the response generated by the OpenAI API. These logs are then evaluated by network or system administrators to find patterns, common attacker behaviour, and for insight gathering. Administrators can also run visualization tools here to understand the given data on a deeper level. Real-time monitoring capabilities, similar to those implemented in Pandora IoT honeypot, [21] are crucial for immediate threat detection and response.

IV. IMPLEMENTATION AND RESULTS

A Deep Q-Network (DQN) framework is utilized, with the state space representing attacker commands, session context, and system parameters. Training is stabilized through experience replay buffers, and Huber loss function handles outliers effectively. This is implemented using Python 3.11.9 and PyTorch is used for the RL model.

Based on insights into attacker behaviour outlined by Salunkhe [22], the commands are categorised and realistic outputs such as fabricated file lists for ls or benign responses for cat are produced.

The LLM Integration, powered by OpenAI's GPT-4o, enhances the honeypot's realism and maximizes attacker engagement.

Similar to the analytical approaches demonstrated by Uppara et al. [23] in intrusion detection systems, our system employs comprehensive logging capabilities to detect patterns in attacker behavior.

To validate our system's performance, we utilized two comprehensive SSH honeypot datasets: the PANDACap SSH Honeypot Dataset by Stamatogiannakis et al., [24]

which contains extensive SSH session logs from real-world attacks, and the CyberLab honeynet dataset by Sedlar et al., [25] which provides a diverse range of attack patterns and behaviours. These established datasets served as benchmarks for comparing GASH's effectiveness in attacker engagement and deception capabilities.

A. Assessment of Performance

A variety of measures such as the duration of the attacker's stay, the quality of the interaction, distribution of actions and gathering of the threat information were some of the factors used to assess the success of the GASH honeypot.

In Fig. 2, the graph shows the evolution of rewards during training over 500 episodes. The various peaks in the reward values indicate successful learning episodes where the model was successful in taking effective decisions and maintaining longer attacker engagements while improving decision-making strategies over time.

Fig. 3 denotes that a window size of 100 episodes is maintained to smooth out the reward trends graph, revealing patterns in the learning progression, with periods of consistent improvement followed by stabilization. The gradual increase in the moving average indicates improving capability to maintain effective attacker engagement.

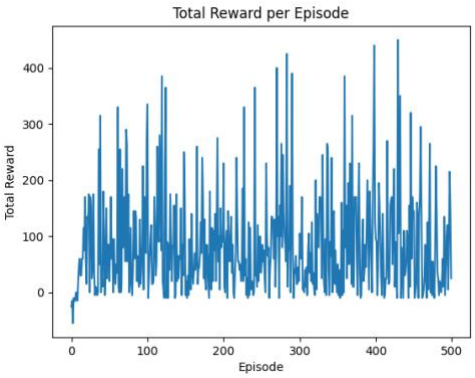

Figure 2. Graphical representation of the total reward per episode versus the number of episodes the RL model is trained on

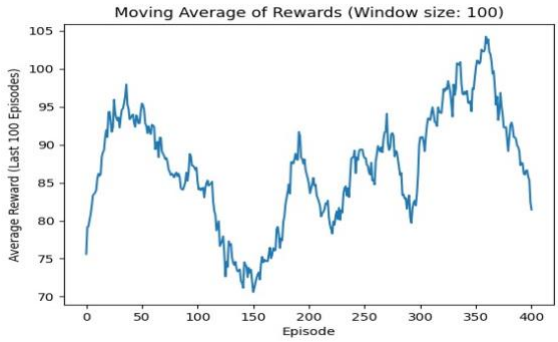

Figure 3. Graphical representation of the moving average of rewards taking a window of size 100 episodes

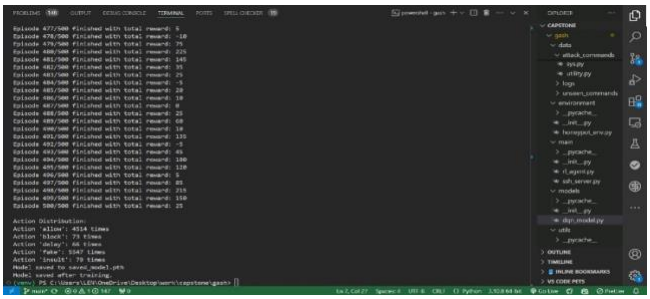

Figure 4. Screenshot denoting the action distribution printed on console after training the model for 500 episodes

In Fig. 4, the console output displays the resultant distribution of the five possible actions (allow, block, delay, fake, and insult) chosen by the DQN agent after running 500 training episodes. This distribution demonstrates the diversified decisions taken by the model to increase attacker engagement, maintaining the honeypot's deception.

B. Analysis of Action Distribution

The DQN agent's distribution of chosen actions showed useful trends in the behaviour of the system. The model dynamically modified its course of action across several sessions in response to interaction patterns and attacker input. While “delay” and “insult” were deliberately employed to irritate attackers without running the danger of instant disconnection, actions like “allow” and “fake” were commonly chosen to preserve realism and continuity. The “block” action was employed sparingly to prevent outright deterrence of attackers while ensuring security constraints were upheld. The results showed that the adaptive nature of the DQN led to a balanced use of actions, ensuring both longer interaction with the system and also effective deception.

C. Threat Intelligence Gathering

The ability of GASH to gather threat intelligence has improved significantly. The system created logs on the behaviour of the attacker by recording all the interactions with the attacker, including inputs, selected actions, and generated answers. These logs were utilised to link behaviours across sessions, find recurring attack trends, and examine the methods attackers used.

V. CONCLUSION

This paper presents GASH, a Generative AI SSH Honeypot that combines reinforcement learning and large language models to overcome the shortcomings of several conventional honeypot systems. GASH enhances attacker engagement and threat information gathering by initialising a Deep Q-Network that makes dynamic choices and uses OpenAI’s GPT-4o to generate replies that are contextually suitable.

Experimental results demonstrate the system’s ability to prolong attacker dwell time. The modular architecture of GASH, incorporating command simulation, and detailed logging, ensures its scalability and robustness and make it an effective tool for getting information on the behaviour of

attackers. This ultimately results in the enhancement of cybersecurity strategy. The results validate that a potential method for creating next-generation honeypots is to combine reinforcement learning with natural language processing.

VI. Future Work

Future iterations of GASH could explore multi-agent reinforcement learning (MARL) to coordinate actions across a network of honeypots. Incorporating adaptive configuration strategies, as explored by Fraunholz et al., [26] could improve GASH's ability to respond dynamically to emerging attack patterns. GASH could also incorporate real-time attacker profiling methods, as proposed by Balogh et al., [27] to enhance the depth and immediacy of threat intelligence analysis. Future versions could also expand the action space and reduce computational overhead associated with DRL and LLM integration.

References

[1] "Federal Information Processing Standards Publication: glossary for computer systems security," Nat. Bur. Stand., FIPS PUB 39, Jan. 1976. doi: 10.6028/nbs.fips.39

[2] Hegde, A.M., Kumar, S.B., Bhuvantej, R., Vyshak, R., Sarasvathi, V. (2023). Spear Phishing Using Machine Learning. In: Singh, M., Tyagi, V., Gupta, P., Flusser, J., Ören, T. (eds) Advances in Computing and Data Sciences. ICACDS 2023. Communications in Computer and Information Science, vol 1848. Springer, Cham. https://doi.org/10.1007/978-3-031-37940-6_43

[3] "2023 Official Cybercrime Report", Cybersecurity Ventures [Online]. Available: https://www.esentire.com/web-native-pages/cybercrime-to-cost-the-world-9-5-trillion-usd-annually-in-2024

[4] DataSoft. (2023). Honeyd: virtual honeypots [Online]. Available: https://github.com/DataSoft/Honeyd

[5] Desaster. (2023). Kippo - SSH Honeypot [Online]. Available: https://github.com/desaster/kippo

[6] Cowrie. (2024). Cowrie SSH/Telnet Honeypot [Online]. Available: https://github.com/cowrie/cowrie

[7] "LaBrea" [Online]. Available: https://labrea.sourceforge.io/labrea-info.html

[8] "thp - Tiny Honeypot" [Online]. Available: http://web.archive.org/web/20090827091208/http:/www.alpinista.org/thp/

[9] G. Wagener, R. State, A. Dulaunoy, and T. Engel, "Heliza: Talking dirty to the attackers," J. Comput. Virol., Aug. 2011

[10] A. Pauna and I. Bica, "RASSH - Reinforced adaptive SSH honeypot," in Proc. IEEE 10th Int. Conf. Commun., May 2014, pp. 1-6.

[11] A. Pauna, A.-C. Iacob, and I. Bica, "QRASSH - a Self-Adaptive SSH honeypot driven by Q-Learning," in Proc. IEEE Int. Conf. Commun., Jun. 2018, pp. 1-6

[12] H. Doubleday, L. Maglaras, and H. Janicke, "SSH Honeypot: Building, Deploying and analysis," Int. J. Adv. Comput. Sci. Appl., vol. 7, no. 5, pp. 1-11, Jan. 2016.

[13] V. S. Mfogo, A. Zemkoho, L. Njilla, M. Nkenlifack, and C. Kamhoua, "AIIPot: Adaptive Intelligent-Interaction Honeypot for IoT Devices," in Proc. IEEE 34th Annu. Int. Symp. Personal, Indoor Mobile Radio Commun., Sept. 2023, pp. 1-6.

[14] B. Wang, Y. Dou, Y. Sang, Y. Zhang, and J. Huang, "IoTCMaL: Towards a hybrid IoT honeypot for capturing and analyzing Malware," in Proc. IEEE Int. Conf. Commun., Jun. 2020, pp. 1-6.

[15] M. Boffa, G. Milan, L. Vassio, I. Drago, M. Mellia, and Z. Ben Houidi, "Towards NLP-based processing of honeypot logs," in Proc. IEEE Int. Symp. Local Metrop. Area Netw., Jun. 2022, pp. 1-6.

[16] J. Buzzio-Garcia, "Creation of a High-Interaction Honeypot System based-on Docker containers," in Proc. IEEE CHILEAN Conf. Electr., Electron. Eng., Inf. Commun. Technol., Jul. 2021, pp. 1-6.

[17] M. Sladić, V. Valeros, C. Catania, and S. Garcia, "LLM in the Shell: Generative Honeypots," arXiv:2309.00155, 2023.

[18] J. S. Lopez–Yepez and A. Fagette, "Increasing attacker engagement on SSH honeypots using semantic embeddings of cyber-attack patterns and deep reinforcement learning," in Proc. IEEE Int. Conf. Secur. Privacy Comput. Commun., Dec. 2022, pp. 1-6.

[19] S. Touch and J.-N. Colin, "A Comparison of an Adaptive Self-Guarded Honeypot with Conventional Honeypots," Appl. Sci., vol. 12, no. 10, p. 5224, May 2022.

[20] M. A. Kristyanto, H. Studiawan, and B. A. Pratomo, "Evaluation of Reinforcement Learning Algorithm on SSH Honeypot," in Proc. Int. Conf. Comput. Sci. Inf. Technol., Dec. 2022, pp. 1-6.

[21] S. Rajesh, V. Paul, V. G. Menon, and M. R. Khosravi, "Pandora: An IOT based Intrusion Detection Honeypot with Real-time Monitoring," in Proc. IEEE Int. Conf. Comput. Intell. Comput. Res., Dec. 2021, pp. 1-6

[22] P. Salunkhe. (2021, May 7). Linux Commands & Utilities Commonly Used by Attackers [Online]. Available: https://www.uptycs.com/blog/threat-research-report-team/linux-commands-and-utilities-commonly-used-by-attackers

[23] Uppara, V., Iqbal, A., P, V., M V, V., V, S. (2024). URL Classification with Intrusion Detection System. In: Arai, K. (eds) Intelligent Systems and Applications. IntelliSys 2023. Lecture Notes in Networks and Systems, vol 822. Springer, Cham. https://doi.org/10.1007/978-3-031-47721-8_19

[24] M. Stamatogiannakis, H. Bos, and P. Groth, "PANDACap SSH Honeypot Dataset," Zenodo, Apr. 2020. doi: 10.5281/zenodo.3759652.

[25] U. Sedlar, M. Kren, L. Š. Južnič, and M. Volk, "CyberLab honeynet dataset," Zenodo, Feb. 2020. doi: 10.5281/zenodo.3687527.

[26] D. Fraunholz, M. Zimmermann, and H. D. Schotten, "An adaptive honeypot configuration, deployment and maintenance strategy," in Proc. Int. Conf. Adv. Comput., Commun. Inform., 2017, pp. 1-6

[27] Á. Balogh, M. Érsok, A. Bánáti, and L. Erdődi, "Concept for real time attacker profiling with honeypots, by skill-based attacker maturity model," in Proc. IEEE 24th Int. Symp. World Wireless, Mobile Multimedia Netw., Jan. 2024, pp. 1-6.