POLITECNICO DI TORINO

Master's Degree Course in Computer Engineering

Master's Degree Thesis

Honeypot and Generative AI

Supervisor
Prof. Andrea ATZENI

Candidate
Enea Gizzarelli

Internship Tutor
Dr. Salvatore PECORARO

Academic year 2023-2024

† To my grandfather, Franco

Summary

The rapidly evolving nature of cybersecurity threats necessitates the development of more adaptive and dynamic defence strategies. Traditional methods often fall short against modern attackers' tactics, amplifying the Defender's Dilemma. While malicious actors only need to exploit one vulnerability, defenders must secure all potential entry points. This thesis addresses this challenge by proposing SYNAPSE (Synthetic AI Pot for Security Enhancement), an innovative approach to cybersecurity defence that combines the deceptive nature of honeypots with the adaptive capabilities of AI. The research explores how such dynamic systems can significantly enhance cyber defences, transforming honeypots from passive lures into active participants in cybersecurity. A vital component of this work is the automatic mapping of logs generated by SYNAPSE to the MITRE ATT&CK framework. By integrating machine learning technologies, SYNAPSE accelerates the identification of attack patterns, offering defenders immediate insights into the strategies employed by attackers.

The Literature Review in Chapter 2 covers foundational concepts surrounding information security, cybersecurity, and honeypot technologies. It also provides an overview of machine learning, generative artificial intelligence and the advancements in large language models (LLMs), which form the backbone of SYNAPSE's AI-driven responses. The literature review also introduces the MITRE ATT&CK framework, establishing its significance in contextualizing cyber threats and mapping SYNAPSE's logs to adversary tactics.

The Methodology, Chapter 3, details SYNAPSE's design, development, and implementation. The system simulates a Linux OS terminal with critical services like SSH and MySQL servers designed to mimic real-world interactions with attackers. In addition, the SYNAPSE-to-MITRE extension is introduced, which automatically maps the collected logs to the MITRE ATT&CK framework. This chapter also includes case studies aimed at evaluating SYNAPSE's functioning. Various tests are implemented, including AI vs AI scenarios where the dynamic honeypot interacts with AI-driven attack tools to assess its adaptability and resilience. DENDRITE, a static equivalent of SYNAPSE, is also introduced to enable comparative analysis, highlighting the advantages of integrating AI into cybersecurity defences.

The Results in Chapter 4 highlight the essential findings from three distinct experiments conducted to evaluate SYNAPSE's performance. The first involved real-world attackers, where SYNAPSE's dynamic interaction capabilities allowed for extracting valuable data. This provided insights into global attack patterns and confirmed the system's effectiveness in capturing critical intelligence. The second experiment involved controlled user interactions by comparing SYNAPSE with DENDRITE, its static counterpart. This analysis emphasized the limitations of traditional honeypots and demonstrated SYNAPSE's superior ability to simulate a realistic environment. Finally, the third experiment focused on AI vs AI scenarios and showcased SYNAPSE's adaptability in responding to AI-driven attacks. It also outlined its ability to accurately map these complex adversarial techniques to the MITRE ATT&CK framework.

This research contributes to the broader field of cybersecurity by showcasing the effectiveness of AI-enhanced honeypots and automated log mapping to MITRE ATT&CK. Future work could explore integrating more advanced AI models, improving operational speed and accuracy, simulating a more comprehensive set of services, and refining its threat mapping capabilities. This thesis lays the foundation for further exploration into AI-driven defence mechanisms, highlighting the importance of dynamic, adaptable tools in an ever-evolving cyber threat landscape.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In cybersecurity, one principle stands indisputable: the field is inherently dynamic and ever-evolving. There's no opportunity to be static; everything needs to change fast to compete with the mutable nature of cyber threats; they continuously morph, adapt, and increase in complexity as new vulnerabilities are discovered and exploited. This exponential evolution means that defensive strategies must also be equally fluid, capable of adapting in real-time to counteract new and unforeseen risks. The Defender's Dilemma is a well-known challenge in the ongoing battle between cyber attackers and defenders. While an attacker needs to find just one vulnerability to exploit, the defender must secure all potential entry points, making defence a far more complex and exhaustive task. An enhanced level of dynamicity in security strategies is essential to address this asymmetry. Static or reactive solutions are no longer sufficient, while effective to a certain extent. They often struggle to keep pace with the rapidly evolving landscape. Traditional cybersecurity technologies can become obsolete almost as soon as they are deployed, undermined by attackers who are always seeking new ways to penetrate them. This dynamic nature requires the implementation of more advanced and flexible defence mechanisms that can evolve with the threats they are designed to counteract.

One critical component of modern cybersecurity strategies is using honeypots. These systems are intentionally deployed to attract attackers, allowing security professionals to monitor and analyze malicious activities and capture their tactics and strategies. They are not static barriers; they are interactive environments for intruders, making them an essential tool for gathering threat intelligence. However, as attackers become more experienced, there is a growing need for honeypots to evolve. Advancements in cybersecurity are occurring alongside the emergence of artificial intelligence (AI), incredibly generative AI, demonstrating significant potential in various fields. It consists of algorithms that generate new content, such as text, images, or code, by learning from existing data. In cybersecurity, generative AI has opened new possibilities for attackers and defenders. Malicious actors may use it to create more convincing phishing attacks or to automate the discovery of vulnerabilities. Defenders can leverage the same technology to enhance security measures.

This is where the intersection of honeypots with generative AI becomes particularly promising. By integrating the adaptive capabilities of AI with the deceptive nature of honeypots, it is possible to create more intelligent, resilient, mutable, and responsive security systems that are harder to detect and evade. Honeypots can be designed to deceive attackers more effectively and learn and evolve in response to the tactics used against them, thereby providing a more proactive defence mechanism in contraposition to the purely reactive one. These dynamic systems will have the power to continuously alter their behaviour and structure to stay one step ahead of attackers, thus alleviating the Defender's Dilemma by making it harder to predict and exploit security measures.

The ability to quickly and accurately interpret data from security systems is paramount in the ever-evolving cybersecurity landscape. Honeypots, which capture extensive logs of attacker activities, generate valuable insights into the methods and tactics employed by adversaries. However, manually analyzing these logs can be time-consuming and prone to oversight, especially given the vast volume and complexity of data. To enhance the effectiveness of honeypots, it is crucial to automatically map the collected logs to the MITRE ATT&CK framework, a comprehensive and widely recognized knowledge base of adversary tactics and techniques. By making this mapping process dynamic, security teams can immediately contextualize the activities observed in honeypot logs, identifying specific attack patterns and understanding the broader strategy behind the intrusion attempts. This accelerates the analysis process and improves threat detection accuracy, enabling defenders to prioritize their responses based on the identified tactics. Furthermore, automatic mapping to the MITRE ATT&CK framework allows for continuously updating defence strategies in line with emerging threats.

The convergence of AI-driven dynamic honeypots with automatic log mapping to frameworks like MITRE ATT&CK gives birth to a new era of evolved cybersecurity. These complex systems can grow by integrating generative AI into honeypots, creating interactions that challenge even the most experienced attackers. This dynamic evolution transforms honeypots from passive lures into active participants in the defence strategy, continuously reshaping the attack surface and introducing unpredictable elements that trick adversaries. Furthermore, automating log mapping through the integration with standardized frameworks like MITRE ATT&CK ensures that these honeypots' vast amounts of data are immediately actionable. This strategic combination accelerates incident response, enabling defenders to recognize patterns, attribute attacks, and refine defensive measures in real-time. As a result, organizations gain a more resilient defence and a more anticipatory security posture that preemptively mitigates risks before they escalate into crises. The defensive paradigm is redefined, shifting the balance of power towards the defenders in the arms race of cybersecurity.

This thesis centres on the development and implementation of SYNAPSE (Synthetic AI Pot for Security Enhancement)[1], a cutting-edge dynamic honeypot designed to address the evolving challenges in cybersecurity. SYNAPSE is a sophisticated honeypot that simulates a Linux OS terminal and leverages the power of generative AI. Unlike traditional honeypots that rely on fundamental terminal interactions, SYNAPSE generates realistic outputs using AI, creating an immersive and deceptive environment for attackers. The current iteration of SYNAPSE includes simulations of two critical services: an SSH server and a MySQL server. These services are engineered to respond to attacker commands as if operating on a natural system, utilizing advanced prompt engineering techniques to enhance realism. Furthermore, SYNAPSE incorporates the SYNAPSE-to-MITRE extension, which automatically maps the logs collected by SYNAPSE to the MITRE ATT&CK framework, leveraging machine learning technologies. By combining the adaptive capabilities of AI with the strategic intelligence of the MITRE ATT&CK framework, SYNAPSE offers a robust and forward-thinking solution to modern cybersecurity challenges, improving both the detection and analysis of cyber threats. This research aims to validate SYNAPSE's effectiveness and explore its potential to transform and enhance the landscape of cybersecurity defence.

1.1 Objectives

The primary objective of this thesis is to explore how the combination of honeypot technology and generative AI can transform cybersecurity defence systems, focusing on the development and evaluation of SYNAPSE, a dynamic AI-powered honeypot. This research will examine how SYNAPSE can evolve traditional honeypots, making them more effective in real-time interactions with attackers and enhancing the overall defence strategy. The thesis also compares SYNAPSE and DENDRITE, a traditional, static system, to highlight the strengths and limitations of incorporating AI into honeypots. The comparative analysis will be essential to evaluate SYNAPSE's ability to simulate a real-world system convincingly while operating in parallel with DENDRITE. This research also assesses SYNAPSE's ability to map its logs to the MITRE ATT&CK framework. By automating this log mapping, SYNAPSE aims to accelerate the identification and classification of attack patterns, offering security teams real-time insights and improving their ability to respond to ongoing threats.

[1] https://github.com/eneagizzarelli/SYNAPSE

Furthermore, the thesis will validate the functionality of SYNAPSE through extensive experimentation, including both controlled scenarios and real-world use cases. One such experiment will involve SYNAPSE interacting with AI-driven attack tools to test its ability to detect, respond to, and mitigate advanced AI-generated threats. In these AI vs AI scenarios, the adaptability and robustness of SYNAPSE will be tested to measure how well it can counteract automated attack strategies and map logs to the MITRE ATT&CK framework. The research will also explore the system's performance in real-world conditions by setting up environments where SYNAPSE engages with genuine attackers or users, providing invaluable insights into its effectiveness under authentic threat conditions.

Ultimately, this thesis aims to contribute to the evolution of cybersecurity defence mechanisms by demonstrating how AI-driven technologies like SYNAPSE can enhance honeypot resilience, allowing for more proactive and dynamic responses to cyber threats. Through real-world experimentation and a comparative analysis with traditional systems, the research seeks to validate the effectiveness of AI-enhanced honeypots and propose innovative strategies for strengthening cybersecurity defences.

1.2 Collaboration with Spike Reply

The successful completion of this thesis was made possible through the collaboration with “Security Reply Srl”, specifically within its “Spike Reply” division, under the “Detection and Response” team. The company provided essential resources and support throughout the research process, enabling the project to be conducted effectively and rigorously. Under the expert supervision of Dr. Davide Manco and Dr. Enrico Cavicchini, the research was guided by their extensive knowledge and commitment to excellence. Their continuous feedback and mentorship ensured that all phases of the project were aligned with academic rigour and industry best practices, enhancing the overall quality and relevance of the work. “Spike Reply” contributed technical and material support and offered a dynamic environment conducive to practical learning and professional development. Participation in the “Detection and Response” team facilitated the integration of advanced cybersecurity tools and methodologies, bridging theoretical knowledge with real-world applications. This collaboration greatly enriched the research, ensuring the outcome was academically sound and practically significant within cybersecurity.

Chapter 2

Literature Review

Before diving into the specifics of SYNAPSE, it is essential to review the existing literature on key concepts such as information security, cybersecurity, honeypots, generative artificial intelligence (AI), machine learning (ML), and the MITRE ecosystem. These areas provide the foundational context for understanding the challenges and innovations that SYNAPSE addresses. By exploring the existing research in these domains, this review will set the stage for a deeper examination of how SYNAPSE builds upon and contributes to these critical areas of study.

2.1 Information Security and Cybersecurity

Security has become a paramount concern for companies. As organisations increasingly rely on digital systems to manage operations, store sensitive data, and interact with customers, the risk of cyber threats has grown exponentially. The consequences of security breaches can be devastating. Beyond the financial impact, the long-term damage to the reputation can be irreparable. Customers and stakeholders expect businesses to safeguard their data, and a failure to do so can lead to a loss of confidence that is difficult to rebuild. Cyberattacks disrupt business operations and expose confidential information, leading to severe regulatory penalties, especially with stringent data protection laws like GDPR. Moreover, the fact that global business ecosystems are connected means that a security breach in one organisation can have a ripple effect, impacting partners, suppliers, and customers.

In this context, security is a fundamental business priority that requires robust security measures, continuous monitoring, and an organisation-wide awareness. Investing in cybersecurity is, therefore, not just about protecting assets; it's about ensuring the business's trustworthiness and future growth in an increasingly digital world. Companies prioritising security can differentiate themselves, demonstrating their commitment to safeguarding their customers' and stakeholders' interests.

2.1.1 Definitions and Comparison

Information Security and Cybersecurity are critical fields that protect assets and sensitive data from unauthorised access, damage, and other malicious activities. While the terms are often used interchangeably, they have distinct scopes (see 2.1). As the National Institute of Standards and Technology (NIST) reports, information security is concerned with "*the protection of information and systems from unauthorised access, use, disclosure, disruption, modification, or destruction to provide confidentiality, integrity, and availability*" [1]. Cybersecurity, a broader term, encompasses protecting systems, networks, and data from cyber threats. We could state that the second is a subset of the first, protecting online or internet-connected devices, systems and technologies from cyberattacks [2].

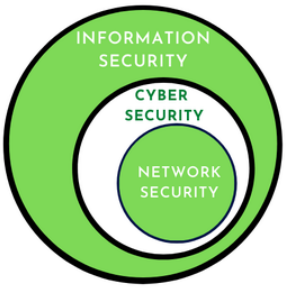


Figure 2.1. Information Security and Cybersecurity (source: GeeksforGeeks).

2.1.2 Core Principles of Information Security

The foundation of information security is built on the CIA triad (see Figure 2.2):

- **Confidentiality**: guarantees that confidential data can only be accessed by authorized individuals, usually through encryption, access controls, and authentication methods.
- **Integrity**: securing data integrity is crucial throughout its lifecycle to prevent unauthorized changes or deletion. This is typically accomplished through checksums, version control, and data validation.
- **Availability**: guaranteeing that authorized users access necessary information and resources at all times demands backup solid solutions, disaster recovery plans, and fault-tolerant systems.

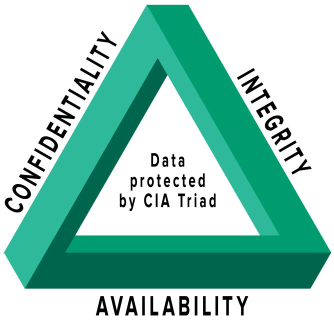


Figure 2.2. The CIA Triad (source: F5 Labs).

2.1.3 The Cybersecurity Threat Landscape

The rapidly evolving threat landscape poses significant challenges to cybersecurity and information security. Threats can be broadly categorised into:

- **Malware**: including viruses, worms, and ransomware, malware disrupts, damages, or gains unauthorised access to information systems.
- **Phishing and Social Engineering**: attackers use deceptive tactics to trick individuals into divulging sensitive information, bypassing technical security measures.

- **Insider Threats**: employees with access to information systems may pose a risk if they misuse their access maliciously or negligently.
- **Advanced Persistent Threats (APTs)**: prolonged and targeted cyberattacks to steal sensitive data or disrupt critical infrastructures, often by nations, states or highly organised criminal groups.

The complexity of these threats necessitates a layered security approach, integrating both technical controls and human factors. Among the innovative defensive strategies being developed, honeypots have gained relevance as an effective way to study attack methods and protect critical assets. The role of honeypots in the modern cybersecurity ecosystem will be discussed in greater detail in the upcoming sections (see Section 2.2).

2.1.4 Emerging Trends in Cybersecurity

As the digital landscape continues to evolve, so does the cybersecurity field, which is increasingly characterised by sophisticated threats and innovative defensive strategies. The dynamic nature of cybersecurity necessitates a continuous adaptation to emerging trends, with a particular focus on integrating advanced technologies, revisiting traditional security principles, and addressing new challenges in the digital realm.

Artificial Intelligence and Machine Learning

One of the most significant emerging trends in cybersecurity is the integration of Artificial Intelligence (AI) and Machine Learning (ML). These technologies have revolutionised threat detection and response by enabling systems to analyse vast amounts of data at unprecedented speeds, identify patterns, and predict potential threats before they materialise. AI-driven tools can automatically detect anomalies, reduce false positives, and prioritise the most critical threats, thereby enhancing the efficiency and effectiveness of cybersecurity efforts. Machine learning algorithms can adapt to new types of attacks by learning from previous incidents, making defence mechanisms more resilient over time. Moreover, the advent of generative AI, a subset of AI focused on creating new content or data, presents both opportunities and challenges in the cybersecurity landscape. In subsequent sections, these aspects will be explored in more detail (see Section 2.3).

Zero Trust Architecture

The traditional perimeter-based security model, which assumed that threats primarily came from outside the network, is increasingly being replaced by the Zero Trust architecture. It is based on the principle that threats can exist inside and outside the network, and thus, no user or device should be trusted by default. This approach requires continuous verification of every access request, regardless of origin. It emphasises the least privilege principle, where users are granted only the minimal access necessary to perform their tasks. By implementing Zero Trust, organisations can better protect against insider threats, lateral movement within the network, and other sophisticated attack strategies.

The Defender's Dilemma

The concept of the defender's dilemma is central to the challenges faced by cybersecurity professionals. This principle highlights the asymmetry in the cybersecurity battle, where defenders must protect all possible entry points, while attackers only need to find a single vulnerability to succeed. This imbalance makes defending networks and systems inherently more difficult as the attack surface expands with the proliferation of devices, cloud services, and remote work environments. The defender's dilemma underscores the need for a proactive and layered security approach, where multiple defensive measures work to minimise the chances of a successful attack.

Security in Depth

Security in Depth is a foundational principle in cybersecurity that emphasises using multiple layers of defence to protect information systems and data. It is critical in addressing the Defender's Dilemma. Implementing multiple layers of defence ensures that even if an attacker penetrates one layer, additional barriers will protect the system, making it less likely that they will succeed. This strategy acknowledges that no security solution is foolproof and that a combination of defences is more effective in mitigating risks.

Future Directions and Evolution

The emerging trends in cybersecurity reflect the growing complexity of the digital world and the need for adaptive, proactive approaches to defence. The integration of AI and ML, the shift to zero-trust architectures, and the recognition of the defender's dilemma are all shaping the future of cybersecurity strategy. As threats evolve, so must the technologies, methodologies, and principles organisations use to protect their data and systems. Staying ahead of these trends is necessary for maintaining security and is critical to modern businesses' broader success and resilience. In the future, we can expect security in-depth to incorporate more advanced measures, further enhancing its effectiveness in protecting against sophisticated cyber threats.

The need for specialised tools and innovative approaches becomes increasingly evident as the cybersecurity landscape evolves. In particular, technologies such as honeypots and generative AI have emerged as crucial components in the arsenal against cyber threats. The subsequent sections of this review will delve deeper into the literature surrounding these technologies, exploring their roles, effectiveness, and the challenges they present in the context of modern cybersecurity (see Sections 2.2, 2.3).

2.2 Honeypot

In an exponentially increasing digital world where malicious attackers can be considered bees ready to sting good actors, a pot of honey able to attract them, consume their time, study them more closely, and extract information about their behaviour could be a precious resource, allowing victims to better prepare for and defend against their attacks. In this analogy, the "pot of honey" represents a strategically designed system or environment miming a vulnerable target, deliberately set up to attract malicious attackers, also known as a "honeypot" in cybersecurity.

A **honeypot** is a decoy computer system whose functioning resides in being attacked or compromised [3], luring cybercriminals in with the promise of an easy target. Once the attackers engage, the honeypot captures detailed information about their methods, tools, and behaviour, producing logs that can be meticulously analysed to understand the nature of the threat. Honeypots are usually an integral part of a layered security approach, often deployed in conjunction with other security measures such as firewalls, intrusion detection systems (IDS), and antivirus software (see Section 2.1.4). This combination enhances the overall defence strategy, with the honeypot acting as both a detection tool and a source of intelligence. In summary, while honeypots are a valuable tool in the cybersecurity arsenal, they are most effective when combined with other security technologies and practices, forming a comprehensive defence-in-depth strategy.

2.2.1 Objectives

The primary objectives of a honeypot are multi-faceted. First of all, they are designed to extract valuable and detailed information about attackers' tactics, techniques, and procedures (TTPs), where

- **tactics** refer to the overall strategy or goal of the attacker,
- **techniques** are the specific methods or approaches used to accomplish these tactics,

- **procedures** are the detailed, step-by-step processes attackers follow to execute their techniques involving specific tools or commands.

By observing and recording the actions of malicious actors in a controlled environment, security teams can be alerted and gain insights into new and emerging threats. This means acting as a general early warning system before critical structures are impacted, also identifying zero-day vulnerabilities that are previously unknown security flaws that attackers might target. Honeypots can lure bad actors into exposing these vulnerabilities before showing them in real-world scenarios. Moreover, they divert attackers from genuine systems and data, delay their progress, and consume their resources. By drawing attention away from tangible assets, honeypots reduce the risk of successful breaches, buying time for the organisation to respond and mitigate potential damage by studying attack tactics in a controlled environment. Indeed, honeypots can be used as a practical tool for training cybersecurity professionals, allowing them to practice detecting and responding to attacks in a supervised setting and test and evaluate the effectiveness of existing security measures. Lastly, captured data can also be used in forensic investigations to trace the origins of an attack, understand its impact, and support legal proceedings against cyber criminals.

2.2.2 Honeynets

A honeynet is a network of honeypots designed to simulate a larger, more complex environment that attracts and studies malicious activity. Unlike a single honeypot, an isolated decoy system, a honeynet consists of multiple interconnected pots representing different network parts. The purpose is to create a more realistic and enticing target for attackers, allowing security teams to observe how they interact with various systems, detect coordinated attacks, and gather comprehensive intelligence on their tactics and behaviours.

While honeynets could provide a more complex and realistic scenario, it is essential to note that they will not be the subject of this thesis. Instead, the focus will remain on the specific applications and benefits of individual honeypots within cybersecurity.

2.2.3 Classifications

Honeypot classification goes hand in hand with the multi-faceted nature of its objectives. Depending on the purpose, deployment environment, and the type of interaction they are designed to have with attackers, honeypots can be categorised in various ways [3]. Understanding these classifications is crucial for selecting the correct type of pot to meet specific security needs, whether the goal is to gather intelligence, detect intrusions, or divert malicious activity away from critical systems. The following subsections will explore the types of honeypots and the criteria used to categorise them, highlighting their unique features and applications (see Table 2.1).

Field of operation

Depending on the context in which they are used, honeypots could be classified as either production or research, each designed to address specific needs.

Production honeypots are typically deployed within live (production) environments to protect genuine systems, of which they will run the same services. Their primary objective is to detect and deflect attacks in real-time, often by diverting malicious traffic away from critical systems. They are more straightforward in design, focusing on minimising risks to the production environment while capturing basic information about threats [4].

In contrast, **research** honeypots are more complex and are primarily used in controlled environments by cybersecurity researchers. They aim to acquire information on new attack techniques, tools, and behavioural patterns. They allow for an in-depth study of malicious methodologies and trends, providing insights that can be used to develop more robust defences. Differently from production ones, they act in the broader world context, contributing to the long-term understanding of emerging threats [4].

Degree of interaction

Honeypots can be categorised based on the level of interaction they allow with attackers, directly impacting the general complexity and the depth of information they can gather. The degree of interaction ranges from low to high, with each level serving different purposes in cybersecurity:

- **Low-interaction**: simulates only a limited set of services, providing minimal interaction to lure in attackers and gather basic information with low risk.
- **Medium-interaction**: offers more complexity, engaging attackers with a broader range of simulated services to collect more detailed data without fully emulating a natural system.
- **High-interaction**: mimics entire systems with their operating environments, allowing attackers to engage deeply. This provides the most comprehensive insights into their methods and tactics but carries a higher risk and complexity.

Type of interaction

Honeypots can be further categorised based on their type of interaction or role within the network, specifically as either server or client honeypots. This distinction is crucial for understanding how these tools interact with attackers and what threats they are designed to attract and analyse.

Server honeypots mimic vulnerable servers or services that attackers commonly target. These honeypots wait passively for attackers to initiate contact, simulating services like web servers, email servers, or databases. The primary objective is to lure attackers who scan for or directly attack these exposed services, capturing their methods and gathering intelligence on the types of exploits they attempt. This is the direction of interaction usually adopted by traditional honeypots.

On the other hand, **client** honeypots operate more proactively. Instead of waiting for attackers to initiate contact, they seek out malicious servers or websites by simulating the behaviour of a typical user or client application. They are often used to detect malicious web content or phishing sites. They are essential for understanding threats that target end-users, such as malware or exploits delivered through compromised websites.

Deployment Environment

Another classification is based on the deployment environment, which is the form in which the honeypot is implemented.

Physical honeypots are hardware devices configured to act as decoy systems within a network. They simulate real machines and are usually fully equipped with an operating system, applications, and services, making them highly realistic targets for attackers. Because they operate on dedicated hardware, they can provide more accurate insights into how attacks might affect natural systems, but they also come with higher costs and complexity.

In contrast, **virtual** honeypots are software-based systems that run on virtual machines (VMs). They are more flexible and cost-effective, allowing multiple instances to be deployed on a single physical server. Virtual honeypots can be quickly scaled, modified, or reset, making them ideal for dynamic and evolving environments.

2.2.4 Placement

Honeypot placement is a critical aspect of network security strategy, designed to lure and monitor malicious activities effectively (see Table 2.2). One of the primary locations for deploying a honeypot is within the **Demilitarized Zone (DMZ)** of a network. The DMZ is a subnet isolated from the internal network, typically exposing public services to the internet. By placing a honeypot in the DMZ, it is possible to attract attackers attempting to access these public

Classification	**Categories**	**Description**
Field of Operation	Production	Deployed in live environments to protect genuine systems by detecting and deflecting attacks in real-time.
	Research	Used in controlled environments for gathering detailed information on attack techniques and trends, often by researchers.
Degree of Interaction	Low-interaction	Simulates limited services, providing minimal interaction to gather basic information with low risk.
	Medium-interaction	Engages attackers with a broader range of simulated services, collecting more detailed data without fully emulating a real system.
	High-interaction	Mimics entire systems, allowing deep attacker engagement and providing comprehensive insights, but carries higher risks and complexity.
Type of Interaction	Server	Mimics vulnerable servers and waits for attackers to initiate contact, capturing methods used against server-based services.
	Client	Proactively seeks out malicious servers or websites by simulating typical user behaviour, useful for detecting threats like malware or phishing targeting end-users.
Deployment Environment	Physical	Hardware devices acting as decoy systems, simulating real machines with operating systems and services, providing accurate but costly and complex insights.
	Virtual	Software-based systems running on virtual machines are more flexible and cost-effective, allowing for easy scaling and deployment in dynamic environments.

Table 2.1. Honeypots Classification

services. This approach allows for monitoring intrusion attempts while minimising the risk to the internal network.

Another strategic location for a honeypot is within the **internal network**, where it attracts and identifies attackers who have already breached the external defences. This is useful for detecting insider threats or identifying lateral movement within the network by providing insights into the techniques and tactics employed by the attackers.

In **wireless networks**, honeypots can be configured as fake wireless access points. These decoys are designed to capture unauthorised access attempts and gather data from nearby devices, thereby enhancing the security of the wireless environment.

Cloud environments also present an opportunity for honeypot deployment. Organisations can attract attacks targeting cloud services and applications by placing a honeypot within a **cloud infrastructure**. This enables the analysis of unique attack techniques in cloud environments, offering valuable information for enhancing cloud security.

Finally, they can be strategically placed near **critical access points**, such as servers, databases, or industrial control systems (ICS). In these scenarios, honeypots serve as early warning tools, providing insights into attempts to access high-value systems and enabling rapid responses to potential threats.

However, when deploying a honeypot, it is crucial to ensure it is completely isolated from the rest of the network. This isolation prevents attackers from using the honeypot as a starting point to launch broader attacks. Additionally, constant monitoring is essential to analyse the collected data and respond swiftly to emerging threats. Without proper isolation and diligent monitoring, the effectiveness of a honeypot can be significantly compromised, potentially introducing new risks rather than mitigating them.

Placement Location	**Description**
Demilitarized Zone (DMZ)	Honeypots placed in the DMZ attract attackers targeting public services exposed to the internet. This allows for monitoring intrusion attempts while minimising risk to the internal network.
Internal Network	Honeypots within the internal network are useful for identifying attackers who have already breached external defences, such as insider threats or lateral movement.
Wireless Networks	Configured as fake wireless access points, honeypots in wireless networks can capture unauthorised access attempts and gather data from nearby devices to enhance wireless security.
Cloud Infrastructure	Honeypots in cloud environments attract attacks targeting cloud services and applications, offering valuable insights into unique attack techniques in the cloud.
Critical Access Points	Placed near high-value systems like servers real-time or industrial control systems (ICS), these honeypots serve as early warning tools for potential threats, enabling rapid responses.
Key Considerations	Honeypots must be isolated from the rest of the network to avoid exploitation. Constant monitoring is essential to analyse collected data and respond to threats in real-time.

Table 2.2. Honeypot Placement Strategies

2.2.5 Legal and Ethical Considerations

When discussing honeypots, we must consider the complex landscape of legal and ethical considerations. While they are powerful tools for detecting and analysing malicious activity, their use raises questions about the legality and morality of specific actions [5].

Honeypots can capture detailed information about attackers, including IP addresses, login attempts, and personal data if compromised accounts or stolen credentials are used. Depending on the jurisdiction, this data could be subject to privacy laws such as the General Data Protection Regulation (GDPR) in the European Union. Organisations must ensure that their honeypot operations comply with them, especially when discussing personal or sensitive information. In addition, honeypots might interact with systems and attackers from various countries. This can raise complex issues as countries have varying laws regarding data collection, cybersecurity practices, and data sharing. Moreover, there's a potential risk that deploying a honeypot could be interpreted as entrapment, mainly if it interacts with attackers in ways that could be seen as encouraging illegal behaviour. If a honeypot is compromised and used to launch attacks on other systems, the organisation could also be liable for damages.

Talking about ethics, we can't help but start with privacy. Honeypots are designed to collect data on malicious actors, but in doing so, they may also capture information from innocent users who inadvertently interact with them. This raises ethical questions about the right to privacy and the extent to which monitoring and recording activities on a network without explicit consent is justifiable. Another issue in this context is the concept of informed consent. Typically, individuals interacting with a honeypot are unaware that their actions are monitored and recorded. This lack of transparency can be viewed as deceptive, leading to questions about whether it is ethical to lure attackers into a trap without their knowledge. The potential for harm is another critical ethical consideration. The consequences can be severe if a honeypot is compromised and used to launch attacks on other systems. This could lead to damage or disruption for third parties and undermine trust in cybersecurity practices if it becomes known that defensive measures like honeypots can inadvertently cause harm. Lastly, there is the question of the proportionality of response. Honeypots, particularly those that are interactive or aggressive, may escalate conflict with attackers. This would provoke more sophisticated or retaliatory attacks, potentially leading to a cycle of escalation that causes more harm than good.

Possible Solutions

Several potential solutions and best practices can be considered to address the legal and ethical concerns associated with honeypot use. These solutions aim to mitigate risks, enhance compliance, and promote ethical standards while still maintaining the utility of honeypots as practical tools for cybersecurity.

One of the most pressing legal concerns is ensuring they comply with data protection laws like the GDPR or other relevant regulations. Honeypots should be designed to collect the minimum amount of personal data necessary for security purposes. Organisations can anonymise or pseudonymise data to protect personal information and reduce the risk of breaching privacy laws. Then, engaging with legal professionals to understand the specific regulatory requirements for honeypots in different jurisdictions is crucial. This helps organisations navigate the complexities of cross-border data handling, ensuring that their operations comply with various laws regarding personal data, cyber activity monitoring, and data sharing. Moreover, honeypot data should be stored only as long as necessary and securely disposed of afterwards. By limiting the retention period, organisations can further protect themselves from compliance risks related to the over-collection of sensitive data.

While honeypots are, by nature, designed to be deceptive, transparency could be improved in certain situations to avoid ethical dilemmas. Organisations could inform network users of ongoing security monitoring for specific internal networks or environments where they might unintentionally capture data from legitimate users. This approach strikes a balance between the need for security and respecting user privacy. Including honeypot use in an organisation's broader privacy and cybersecurity policies can help set expectations. Transparency in the intent to use advanced threat detection methods like honeypots can mitigate the ethical concerns related to deception.

Organisations can limit the interactivity of their honeypots to avoid the risk of entrapment or provoking escalatory actions. Low-interaction ones will be less likely to engage attackers in ways construed as encouraging illegal behaviour. Interactive ones should not communicate directly with malicious actors in ways that could be interpreted as luring them into more complex offences. They should focus on passive observation and data collection. Honeypots should be part of a broader cybersecurity strategy that includes firewalls, intrusion detection systems (IDS), and other non-deceptive tools, diversifying security methods.

To address ethical concerns, the cybersecurity community could benefit from clear moral guidelines for honeypot deployment. Organisations can follow established ethical frameworks to ensure their actions align with societal expectations. Adopting the principles of respected ethical guidelines could help assess whether using honeypots is responsible and proportionate to the threats to mitigate. Large organisations or those that frequently use these systems could establish internal ethics committees to review and assess the implications of their cybersecurity practices. Such committees would ensure that the honeypot's use is justified and that the potential harm to third parties or innocent users is minimised.

A key ethical issue is the potential to harm third parties if the system is compromised. Honeypots should be set up so they cannot be easily used as launching points for further attacks. They should be isolated from an organisation's core systems to limit the damage if they are compromised. Honeypots should be regularly updated and monitored to ensure attackers do not exploit them to launch new cyber threats. Deploying pots with outdated or known vulnerabilities can increase the risk of harm, undermining their utility as a security tool. A rapid incident response plan can mitigate damage if one is compromised. Organisations should have automated triggers that deactivate the tool or alert security personnel when unusual activity is detected.

The cross-border nature of cyberattacks often makes it challenging to manage the legal implications of honeypot use, especially in cases involving multiple jurisdictions. The international collaboration between governments and cybersecurity agencies can help harmonise laws and policies concerning deployment. Promoting and adhering to international standards such as those proposed by the International Organization for Standardization (ISO) can help create a unified approach to managing the legal risks of cybersecurity tools, including honeypots. Engaging law enforcement agencies during the planning and deployment can help ensure their use complies with legal norms. Cooperation with authorities can also be crucial in cases where honeypots uncover evidence of significant criminal activity.

2.2.6 Real-World Examples

To better understand the practical applications and effectiveness of honeypots, this subsection proposes real-world examples implemented in various contexts.

The Honeynet Project

One of the most well-known examples is "The Honeynet Project"[1], a non-profit global research organisation dedicated to improving cybersecurity by providing resources and tools for deploying honeypots [6]. Established in 1999, the Honeynet Project has developed various open-source tools that organisations and researchers use worldwide. Their technologies, such as Honeyd and Dionaea, are designed to emulate vulnerable systems and capture a wide range of attack data. The project has been fundamental in advancing the understanding of cyber threats and has contributed significantly to developing new security strategies.

Honeyd

"Honeyd"[2] is a low-interaction honeypot that allows users to create virtual hosts on a network, simulating various operating systems and services. It helps researchers observe how attackers probe and exploit different vulnerabilities [7]. By emulating different network topologies, Honeyd enables the detection of scanning techniques, unauthorised access attempts, and specific attack patterns targeting emulated services. Its flexibility and wide adoption have made it a foundational tool for understanding common network attacks.

[1] https://www.honeynet.org/

[2] https://www.honeyd.org/

Figure 2.3. The Honeynet Project (source: The Honeynet Project - Github).

Dionaea

"Dionaea"[3] is a medium-interaction honeypot designed to capture malware. It mimics vulnerable services to attract and capture exploits and malicious binaries that spread through vulnerabilities in standard network services [8]. Dionaea's ability to collect and categorise malware samples makes it a valuable tool for cybersecurity researchers studying evolving tactics, techniques, and procedures (TTPs). It has been instrumental in generating actionable threat intelligence by providing insights into new malware variants and their propagation methods.

Cowrie

"Cowrie"[4] is a popular medium to high interaction honeypot designed to simulate SSH and Telnet services. It is widely used to detect and analyse brute-force attacks and capture data on the commands executed by attackers once they gain access [9]. Cowrie logs all interactions in detail, providing valuable insights into the behaviour and methods of attackers targeting SSH-enabled devices. Many organisations and researchers use Cowrie to monitor and understand SSH-based attacks, making it a useful tool for threat intelligence.

2.3 Generative Artificial Intelligence (AI)

The advent of **Artificial Intelligence (AI)** marks a pivotal shift in how technology interacts with the world around us. At its essence, AI refers to the "*technology that enables computers and machines to simulate human learning, comprehension, problem-solving, decision making, creativity and autonomy*", standing to the definition proposed by IBM [10]. As AI has evolved, it has transcended the concept of traditional computational systems, creating new paradigms, automation, and problem-solving that permeate every aspect of modern life, from industry and healthcare to entertainment and communication. AI is not a singular entity or technology but a broad spectrum of methodologies and approaches that enable machines to simulate human intelligence (see Figure 2.4). Its foundation lies in processing large datasets, identifying patterns, and adapting to new information, allowing systems to make decisions and improve their performance over time. The profound capabilities of AI have redefined industries, enabling everything from autonomous vehicles to personalised medicine to more intelligent infrastructure.

Within this exponentially growing field, **Generative AI** has emerged as a particularly innovative branch, built upon the fundamental principles of artificial intelligence. Unlike traditional AI models that primarily analyse, classify, or predict based on existing information, generative AI

[3] https://github.com/DinoTools/dionaea

[4] https://github.com/cowrie/cowrie

can generate new data almost indistinguishable from human-produced outputs [11]. Learning underlying patterns and structures can produce novel outputs that mimic real-world data, whether generating human-like text, images, videos, or even complex simulations.

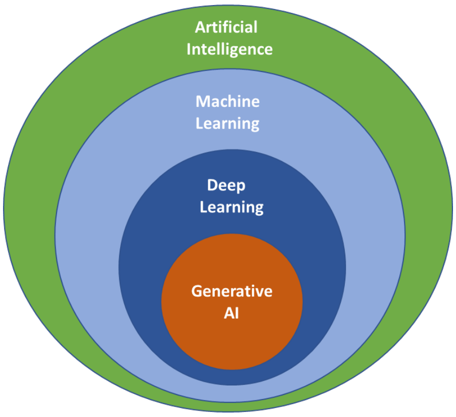


Figure 2.4. AI landscape (source: Medium).

2.3.1 Large Language Models (LLMs)

Large Language Models (LLMs) represent a significant advancement within the broader field of artificial intelligence. These models are designed to understand, generate, and manipulate human language by training on vast amounts of text data powered by deep learning architectures. They process language, catching its intricacies such as grammar, semantics, and context-dependent meanings. LLMs develop a sophisticated understanding of linguistic patterns by being pre-trained on massive datasets, often comprising billions of words. Fine-tuning (see 2.3.1) these models enable them to perform specific tasks such as translation, summarisation, question-answering, and even creative writing [12].

The relationship between LLMs and generative AI is particularly profound, as LLMs represent one of its most powerful applications. LLMs excel at generating human-like text. For instance, models like GPT-3 and GPT-4 use their training on extensive datasets to predict and create paragraphs or pages of coherent, creative text. This capability is critical to applications such as automated content creation, real-time conversation systems, and personalised language-based services [13]. Moreover, LLMs rely on the transformer architecture, a foundational innovation in generative AI, allowing them to process language in parallel and understand dependencies within text. This architecture significantly enhances their ability to produce contextually aware and semantically rich outputs, which is critical for dialogue generation, machine translation, and creative writing. The ability of LLMs to generate not only grammatically correct but also context-sensitive and nuanced text exemplifies their central role in advancing generative AI, aligning them closely to simulate or enhance human creativity and intelligence.

Fine-Tuning

Fine-tuning is a crucial process in training Large Language Models (LLMs) that allows them to specialise in specific tasks. The model is first pre-trained on a large dataset that covers a wide range of topics and language uses but is still not task-specific. A dataset relevant to the specific task or domain is curated for fine-tuning. It is typically much smaller than the one used in pre-training but is closely aligned with the intended use case. For example, the dataset would consist of labelled text with positive, neutral, and negative sentiments for a sentiment analysis model. Later, the pre-trained model is further trained on this task-specific dataset. Performance is then evaluated on a validation set to ensure the adjustments improve the overall result. After fine-tuning, the model is better suited for the specialised task and can be deployed in applications requiring task-specific language generation or understanding.

Tokenisation

Tokenisation is breaking down a text stream into smaller, more manageable units called **tokens** [14]. These are essential because LLMs cannot process raw text directly; instead, they must be presented numerically, such as vectors. Tokenisation transforms words, subwords, or even individual characters into tokens, which the model then uses to understand and generate language. The input to the tokeniser is a string of text, such as a sentence or paragraph. The tool breaks down the text into units whose granularity depends on the tokeniser type. Each token is later assigned a unique integer identifier from the model's vocabulary. After tokenisation, the integer tokens are transformed into vectors of numbers, "embeddings", which capture the semantic relationships between them. These are then passed to the LLM, enabling the model to process and generate coherent text.

State of the Art

Large Language Models (LLMs) have seen rapid advancement, with various cutting-edge models being developed to push the boundaries of natural language understanding, generation, and interaction. These models are characterised by their impressive scale in parameters and training data and their capacity to generate highly coherent and contextually aware outputs. In this section, we will explore five of the leading LLMs currently shaping the AI landscape (see Table 2.3):

- **GPT (Generative Pre-trained Transformer)**[5]: GPT, developed by OpenAI, remains one of the most widely recognised and powerful LLMs available. Initially introduced with GPT-2 and later expanded with GPT-3 and GPT-4, this series has set the standard for text generation capabilities. GPT-4 introduced improved contextual understanding and the ability to handle more complex prompts. The model has found applications in various areas, from creative writing to professional tasks like coding and data analysis. This is an honourable mention to GPT-4o, a recent model optimised for faster and more efficient operations while maintaining high-quality text generation, making it highly suitable for real-time applications.
- **Llama (Large Language Model Meta AI)**[6]: Llama, developed by Meta AI, is designed to provide high performance with a focus on accessibility and efficiency. Llama 3.1, with a staggering 405 billion parameters, is among the latest advancements in this model series. Honourable mention to Llama 3.1 405B, one of the most significant models in terms of parameters, focusing on efficient performance at scale.
- **Claude**[7]: Claude, developed by Anthropic, is known for its focus on safety and alignment in AI systems. The Claude series of LLMs emphasises creating powerful and robust models to ensure the ethical use of AI. The Claude Sonnet 3.5 release represents the latest iteration, an advanced version emphasising safety and alignment in AI, with improved task handling.
- **Gemini**[8]: Gemini, developed by Google DeepMind, is another state-of-the-art LLM focused on multimodal capabilities, meaning it integrates both text and image processing in a unified framework. Honourable mention to Gemini 1.5 Pro, which brings an enhanced processing power and better integration between textual and visual data, improving its ability to generate more accurate and relevant outputs across multimodal tasks.
- **Grok**[9]: Grok, developed by xAI, is a newcomer to the LLM scene but quickly gaining attention. Grok 2, the latest release, enhances context handling and adaptability, making it suitable for various innovative and complex tasks.

[5] https://platform.openai.com/docs/models

[6] https://llama.meta.com/

[7] https://docs.anthropic.com/en/docs/about-claude/models

[8] https://deepmind.google/technologies/gemini/

[9] https://x.ai/blog/grok-2

LLM	Developer	Recent Release	Key Features
GPT	OpenAI	GPT-4o	Optimized for speed, efficiency, and complex tasks.
Llama	Meta AI	Llama 3.1 (405B)	405B parameters, efficient performance, scalability.
Claude	Anthropic	Claude Sonnet 3.5	Safety-focused, improved task handling, ethical AI.
Gemini	Google DeepMind	Gemini 1.5 Pro	Enhanced multimodal capabilities (text and images).
Grok	xAI	Grok 2	Improved contextual depth and creative adaptability.

Table 2.3. Summary of Leading LLMs and Their Recent Releases (source: lmarena)

2.3.2 Prompt Engineering

Every time we engage with a generative AI model, such as ChatGPT[10], through a browser or application, we are unwittingly performing a task central to the functionality of these systems: writing a prompt. This simple input, a question, a statement, or a command, sets off a complex chain of computational processes. The AI interprets the text, drawing on its vast database of learned information, and produces an output tailored to the prompt. However, inputting text does not guarantee the output's quality, relevance, and usefulness. Instead, it depends on the design of the prompt itself. For example, asking a model a vague or open-ended question may bring an equally ambiguous answer, while a carefully constructed prompt, which is clear and specific, can generate a highly accurate and contextually appropriate response. This introduces the concept of **prompt engineering**, the deliberate and strategic crafting of prompts to guide AI in producing the desired outcome.

Prompt engineering is the art and science of developing and optimising inputs to achieve precise, relevant, and reliable outputs from generative models [15]. Much like instructing a human to complete a task, the more detailed and unambiguous the instructions, the better the result. By following specific prompt engineering techniques, users can manipulate the model's outputs to be more coherent, focused, and aligned with their expectations, making this practice indispensable for leveraging the full potential of generative AI.

Techniques

Optimising the interaction with generative AI models requires various techniques that help users craft prompts for more accurate and targeted outputs. These techniques are designed to minimise ambiguity, enhance the relevance of the response, and better control the AI's behaviour. Some of the most effective prompt engineering techniques include [15]:

- **Zero-Shot Prompting**: refers to the ability of a generative AI model to perform a task without being explicitly provided with any prior examples. The AI relies entirely on the knowledge encoded within its training data in this method. The prompt directly describes the desired outcome, and the model generates a response based on its internal understanding of the task. For example, suppose a user asks, "Summarise the main points of this thesis". In that case, the model is expected to understand the concept of summarisation and perform the task without seeing examples of previous summaries. Zero-shot prompting is particularly powerful because it enables the AI to generalise across various tasks without specific training. However, its effectiveness depends on the clarity of the prompt and the model's ability to understand the task based on its training data. The output might be less accurate or coherent if the task is novel or complex.

[10] https://openai.com/chatgpt/

- **Few-Shot Prompting**: technique where users provide the AI with a few examples to guide the model's understanding of the task. Unlike zero-shot prompting, few-shot prompts give the AI explicit guidance, allowing it to mimic the provided examples' structure, style, or content. For instance, if the task is to generate customer service email responses, a few-shot prompt might include two or three example emails followed by a new request. The model uses the examples provided to understand the pattern and generate a response that follows the same format. This approach significantly improves the model's ability to handle specialised or nuanced tasks because it reduces ambiguity and helps align the AI's output with user expectations.

- **Chain-of-Thought Prompting**: an advanced technique that guides the AI to generate a step-by-step explanation of its reasoning process before arriving at a final output. Instead of asking the model for a direct answer, the user prompts the AI to articulate its intermediate reasoning stages, simulating human logical processes. For example, when solving a math problem, the prompt might be: "First, describe how you would approach solving this equation. Then, show the steps before providing the solution". This structured reasoning helps the model focus on the logic required to solve complex or multi-step tasks, reducing the likelihood of errors in reasoning. Chain-of-thought prompting is especially valuable when the task demands more than a superficial response, such as in mathematical logic, logical deduction, or strategic decision-making. The model can address each aspect more accurately by breaking the task into smaller parts, often leading to more robust and correct final outputs. This technique also improves interpretability, allowing users to understand how the AI arrived at its conclusions.

- **Persona Prompting**: this method explicitly defines a role for the AI to adopt, guiding its responses following the characteristics, behaviour, and style associated with that role. For instance, when a user instructs the model with a prompt such as, "You must act as a Linux terminal and respond exactly as such," the AI behaves consistent with a Linux terminal's functionality and output style. This approach allows users to simulate specific environments or contexts, such as technical systems, professional personas, or instructional settings. By establishing clear guidelines for how the AI should behave, this technique enhances the relevance and accuracy of the generated content, making it more aligned with the user's specific needs and expectations.

2.3.3 Limitations

Despite the significant advancements in generative AI, several limitations hinder its performance and application in various fields. These limitations, inherent in generative models' architecture and training processes, raise concerns about their reliability, ethical implications, and scalability. First, generative AI models do not understand the world as humans do; instead, they produce outputs based on probabilistic associations between words, phrases, or patterns in data. This lack of proper understanding can lead to significant inaccuracies, commonly called "hallucinations". These occur when the model generates incorrect or wholly fabricated information. For instance, a language model may provide a plausible answer but is factually wrong. This limitation becomes particularly problematic in fields where precision is critical, such as medicine, law, or science, where incorrect outputs can have serious consequences. The model's inability to verify or cross-reference its responses further exacerbates this issue, highlighting the need for human oversight in AI-generated content.

The rise of generative AI has diffused ethical concerns, particularly around the potential misuse of these technologies. AI-generated content can be exploited to create deepfakes, realistic but fake videos, or audio recordings that can be used for disinformation or defamation. Additionally, generative AI can be leveraged to produce fake news or manipulate public opinion, leading to the spread of misinformation at an unprecedented scale. These ethical concerns pose significant challenges for regulation, as there is no universally accepted framework for addressing the risks posed by AI-generated content. Balancing innovation with responsible use is a critical issue that remains unresolved as generative AI technologies advance. Another ethical concern involves the amplification of biases. Generative AI models are trained on vast datasets, which often contain underlying biases reflective of societal inequalities. These models can unintentionally reproduce and even amplify discriminatory content related to race, gender, and other identity factors. This poses a serious ethical dilemma, mainly when AI is used for decision-making where biased outcomes can perpetuate inequalities. Developers and organisations must carefully curate training data and implement fairness measures to mitigate these risks, though eliminating bias remains a significant challenge [16]. Ownership and intellectual property (IP) rights present further complications. Generative AI models can produce original text, music, or art content, blurring the lines between human and machine creativity. This raises questions about who owns the rights to AI-generated works. Current copyright laws, which are designed for human creators, may not adequately address the issue of machine-generated content. If an AI creates a work based on data it was trained on, should the creators of the original datasets be entitled to some form of acknowledgement? This grey area in IP law is still largely unexplored, and new legal frameworks are needed to address the ownership of AI-generated intellectual property. Finally, concerns around accountability and transparency are critical. If an AI model generates harmful or misleading content, it can be difficult to assign responsibility. Should the creators of the AI system be held liable, or should the users who deploy it have the consequences? The opacity of many AI models, often called "black-box" systems, further complicates the issue. These systems are so complex that even their developers may not fully understand how specific outputs are generated, making accountability challenging to enforce.

Among the other limitations, developing, training, and running large-scale generative AI models requires immense computational resources. These models, particularly those with billions of parameters, such as GPT-3 or its successors, demand substantial energy, processing power, and time to train. This creates accessibility barriers for smaller companies, organisations, or individuals who lack the necessary infrastructure to develop or deploy generative AI on a large scale. Additionally, the environmental impact of these resource-intensive processes raises concerns about the sustainability of generative AI, especially as the demand for increasingly larger models continues to grow [16].

Generative AI models often also struggle with "overfitting" where they become too attuned to the training data and fail to generalise effectively to new, unseen inputs. This limitation hampers their ability to produce novel or creative outputs, as they tend to replicate the patterns learned from training data rather than generate original content. The challenge lies in balancing the ability to memorise training data with the need to generalise well to diverse inputs. Achieving this balance is essential for improving the model's creativity and utility in real-world applications with dynamic conditions, where data can vary significantly.

Despite their capabilities, generative AI models often require significant human supervision, mainly when applied in sensitive or high-stakes domains. Human intervention is necessary to correct errors, ensure outputs are factually accurate, and avoid unintended ethical or social consequences. This reliance on human oversight limits the extent to which AI systems can be fully autonomous, as users must constantly monitor and evaluate the outputs generated. Additionally, this supervision requirement reduces the efficiency gains promised by AI, as human operators are still needed to guide the systems, particularly in complex or critical scenarios.

2.4 Machine Learning (ML)

Machine learning (ML) is a subfield of artificial intelligence (see Figure 2.4) that focuses on developing algorithms that allow computers to learn from problem-specific training data and make decisions or predictions without being explicitly programmed, automating the process of model building to solve associated tasks [17]. ML identifies patterns and structures within vast datasets to make informed decisions. This process mimics the human ability to learn from experience and improve decision-making over time. As a critical component of modern AI, machine learning has evolved rapidly over the past decades, influencing various industries, including finance, healthcare, marketing, and cybersecurity. Also, it has become foundational in developing generative models, such as those used in generative AI systems (see Section 2.3.1).

2.4.1 Types

The learning process in machine learning is often categorised into three main types, where each one is designed to solve specific problems and operates differently based on the nature of the available data and desired outcomes:

- **Supervised Learning**: involves training a model on a labelled dataset, where the input and the corresponding output are known. The model aims to generalise from the training data and predict new, unseen data outcomes. Once the model has been successfully trained, it can indicate a target variable given the input features' new or unseen data points [17].
- **Unsupervised Learning**: works with data that has no labels. The aim is to uncover hidden patterns or intrinsic structures within the data. Thus, the training set only consists of variables to find structural information of interest, such as groups of elements that share common properties, also known as "clustering" [17].
- **Reinforcement Learning**: the model learns through trial and error, receiving feedback as rewards or penalties based on the actions it takes in a given environment. This is particularly useful in robotics or game AI tasks, where decisions must be made sequentially.

2.4.2 Training of a Model

Training a machine learning model is a multi-step process that transforms raw data into actionable insights, allowing the model to learn from patterns in the data. The ultimate goal is for the model to generalise from the training information to make accurate predictions on unseen data.

Data Collection and Preprocessing

The first step in training any machine learning model is acquiring and preparing the dataset. The latter must be relevant to the problem and represent the diversity of scenarios the model will encounter in the real world. Raw data often contains noise, missing values, or inconsistencies in real-world applications, which can negatively impact the model's performance if mishandled. For this reason, data preprocessing involves several vital tasks:

- **Cleaning the data**: removing or imputing missing values, handling outliers, and ensuring consistency in the dataset.
- **Normalization or standardization**: bringing features onto a common scale. This is especially important when using distance-based algorithms or neural networks like MLP, where feature values of different scales can skew the model's learning process.
- **Feature engineering**: creating new features or transforming existing ones to enhance the model's ability to learn important patterns. For example, combining several weak features into a single, more meaningful one.

Once preprocessing is complete, the dataset is typically split into training, validation, and test sets. The training set is used to fit the model, the validation set helps tun the model's hyperparameters, and the test set evaluates the model's performance on unseen data.

Model Training Process

Training a machine learning model involves feeding data through it and iteratively adjusting parameters to minimise the difference between the predicted output and the actual outcome. At the beginning of training, the model is initialised with random values for its parameters (e.g., weights in the case of neural networks). These parameters will be updated iteratively to improve the predictions. In each iteration, or "epoch", the training data is passed through the model in a forward-pass process, producing a prediction. The model's architecture, whether a simple decision tree, a linear model, or a neural network like MLP, dictates how these predictions are made.

The next step is calculating how far off these predictions are from the target values. This is done using a loss function, which quantifies the error. After calculating the loss, the model's parameters must be updated to minimise it. In neural networks, including MLP, this is done through "backpropagation", which calculates the loss gradient concerning each parameter. Then, an optimisation algorithm updates the model's weights in the direction that reduces the loss. Forwarding the input, calculating the loss, and updating the weights is repeated over multiple epochs. The model is said to "converge" when the loss no longer decreases significantly, indicating that it has learned the essential patterns in the training data.

Algorithms

When training a machine learning model, the choice of algorithm plays a crucial role in making predictions accurately. Different algorithms are suited to other tasks, each with strengths and weaknesses. In this section, it is possible to find some of the most relevant examples:

- **Decision Trees**: versatile models that split data into subsets based on feature values to form a tree-like structure. They work well for classification and regression tasks and are easy to interpret. However, decision trees can overfit the training data if not carefully controlled.
- **Random Forest**: ensemble learning method that improves upon decision trees by creating multiple trees and averaging their predictions. Combining the outputs of various trees reduces overfitting and generally performs better than a single one. Random forests are robust and effective on multiple tasks but can become computationally expensive as the number of trees grows.
- **K-Nearest Neighbours (KNN)**: classifies data points by looking at the nearest neighbours in the training data. It's simple and non-parametric, making no assumptions about the underlying data distribution. KNN can work well for small datasets but becomes inefficient and less effective as the dataset grows or becomes high-dimensional.
- **Naive Bayes**: a probabilistic classifier based on Bayes' Theorem, assuming that features are independent given the class label, hence "naive". Despite this simplifying assumption, Naive Bayes can perform surprisingly well for specific tasks, particularly in text classification. However, it may struggle when the independence assumption between features does not hold, leading to reduced accuracy.
- **Multi-Layer Perceptron (MLP)**: a type of neural network well-suited for handling complex, non-linear relationships. MLP consists of an input, hidden, and output layer. Each layer of neurons transforms the input through learned weights and activation functions. MLP is particularly powerful in classification tasks where simpler models like decision trees may fail, as it can model complex interactions between features. Training an MLP involves forward propagation, where input data is passed through the network to produce a prediction, followed by backpropagation, where errors are propagated backwards through the network to adjust the weights. This iterative process allows the model to improve over time. While MLP is highly flexible and powerful, it requires more computational resources and training data than simpler models.

2.4.3 Classification

Classification is one of the core tasks in machine learning, where the objective is to assign input data to predefined categories or labels. It is a form of supervised learning (see Section 2.4.1), meaning that the model is trained using labelled data, where the correct category for each input is known. The model learns from these labelled examples and then generalises this knowledge to classify new, unseen data.

In a typical classification problem, the dataset consists of input features, like object or scenario characteristics, and corresponding labels, such as categories like “spam” or “not spam”. The model is trained on a labelled dataset where it observes input-output pairs. During this phase, it learns the patterns and relationships between the input features and the corresponding labels. Once trained, the model can classify new data points by assigning the label that best fits the learned patterns. This is done by using the learned weights or rules to compute the probability or score for each class, later assigning the one with the highest score.

Performance Evaluation

Various evaluation metrics measure how effectively the model assigns data points to the correct categories to ensure the classification performs well. The most common metrics include:

- **Accuracy**: the most straightforward metric representing the ratio of correct predictions (true positives and negatives) to the total number of predictions.

  $$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

  Where:

  - TP = True Positives (correct positive predictions)
  - TN = True Negatives (correct negative predictions)
  - FP = False Positives (incorrect positive predictions)
  - FN = False Negatives (incorrect negative predictions)

  Although accuracy is widely used, it can be misleading if the dataset is imbalanced, meaning one class is significantly more frequent than the other.

- **Precision**: it measures the proportion of correctly predicted positive instances out of all the cases predicted as positive. It is beneficial when the cost of false positives is high.

  $$\text{Precision} = \frac{TP}{TP + FP}$$

  A high precision means that it is usually correct when the model predicts a positive class.

- **Recall**: also known as “sensitivity” or “true positive rate”, measures the proportion of actual positive instances correctly identified by the model. It is relevant when false negatives, such as medical diagnoses, are costly.

  $$\text{Recall} = \frac{TP}{TP + FN}$$

  High recall indicates that the model can identify most of the actual positives.

- **F1 score**: the harmonic mean of precision and recall, providing a balanced measure between the two. It is useful when an uneven class distribution or precision and recall are essential.

  $$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

  The F1 score ranges from 0 to 1, with 1 being the best possible value.

- **Confusion Matrix**: it provides a more detailed view of the model’s classification performance by showing the counts of true positives, true negatives, false positives, and false negatives:

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

  The confusion matrix is handy for analysing the errors a model makes (e.g., false positives vs. false negatives).

Multi-Class Classification

These metrics can be extended to multi-class classification, where multiple categories are predicted instead of binary classes. Metrics like precision, recall, and F1 score can be computed for each class independently and then averaged using either "macro-averaging" (treating each class equally) or "weighted averaging" (giving more importance to frequent classes). Similarly, a "multi-class confusion matrix" expands to include each class's counts of true positives, false positives, and false negatives, providing a comprehensive view of classification performance.

2.4.4 Natural Language Processing (NLP)

Natural Language Processing (NLP) is a subfield of artificial intelligence (AI) focused on the interaction between computers and human language. It enables machines to understand, interpret, and generate human language meaningfully and helpfully. It involves several critical tasks, including:

- **Tokenisation**: the process of splitting text into smaller units, such as words or phrases, for easier processing (see Section 2.3.1).
- **Part-of-Speech Tagging (POS)**: assigning parts of speech, such as nouns, verbs, and adjectives, to each word in a sentence to better understand its grammatical structure.
- **Named Entity Recognition (NER)**: identifying and classifying entities such as people, places, and organizations within the text.
- **Sentence or Text Classification**: categorizing entire sentences or documents into predefined categories, such as spam detection in emails or topic classification in news articles.
- **Sentiment Analysis**: determining the sentiment or emotional tone behind a piece of text, whether positive, negative, or neutral.
- **Language Modeling**: predicting the next word in a sentence based on the previous words is crucial for applications like autocomplete and text generation.

Recent advancements in NLP have been driven by deep learning models, particularly transformer-based architectures like BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer). These models have significantly improved the ability of machines to understand context and generate human-like text by leveraging large-scale datasets and unsupervised learning techniques. NLP is particularly impactful in fields such as customer service, healthcare, and marketing, where it automates text analysis, enhances customer interactions through chatbots, and provides valuable insights through sentiment and trend analysis.

2.5 MITRE ATT&CK

The **MITRE Corporation** is a not-for-profit organisation that operates federally funded research and development centres (FFRDCs) in the United States. Established in 1958, MITRE's mission is to tackle some of the most complex problems facing government agencies and other institutions, particularly national security, defence, and cybersecurity. Over the decades, MITRE has become a pivotal player in the cybersecurity landscape, known for its rigorous research, innovative solutions, and comprehensive frameworks that guide cybersecurity professionals in understanding, detecting, and mitigating various threats. MITRE's role in security is multi-faceted, from developing standards and methodologies to collaborating with public and private entities on cybersecurity initiatives. The organisation's approach is characterised by its emphasis on creating practical tools and frameworks that security teams can widely adopt to enhance their defence mechanisms [18].

One of MITRE's most influential contributions to the cybersecurity field is the development of the **MITRE ATT&CK**[11] (Adversarial Tactics, Techniques, and Common Knowledge) framework, "*a globally-accessible knowledge base of adversary tactics and techniques based on real-world observations. With the creation of ATT&CK, MITRE is fulfilling its mission to solve problems for a safer world - by bringing communities together to develop more effective cybersecurity.*" [19]. This framework has become a globally recognised cornerstone for cybersecurity strategies. ATT&CK was initially developed to document the tactics and techniques used by threat actors as observed in real-world cyber incidents. Over the years, it has evolved into a comprehensive knowledge base that helps organisations understand how attackers operate, allowing them to better anticipate and defend against potential threats.

2.5.1 Matrices

The MITRE ATT&CK framework is organised into three distinct **technology domains** or matrices[12], each representing the ecosystems in which adversaries operate [20], with the possibility of having multiple platforms or operating systems for each domain.

- **Enterprise**: this domain covers a wide range of platforms, including operating systems such as Windows, Linux, and macOS, as well as cloud environments like Azure AD, Office 365, Google Workspace, Software-as-a-Service (SaaS), and Infrastructure-as-a-Service (IaaS). It also includes network infrastructure devices and container technologies. See Figure 2.5 for a visual example of the matrix.
- **Mobile**: this domain models adversarial tactics and techniques specific to Android and iOS platforms. It includes a separate matrix outlining techniques an adversary can use without direct access to the mobile device.
- **Industrial Control Systems (ICS)**: This domain is dedicated to adversarial tactics and techniques to disrupt industrial control processes.

These domains provide tailored frameworks for different environments, helping defenders understand and mitigate specific threats in each context.

2.5.2 Levels

As CISA (.gov) reports in its publication "Best Practices for MITRE ATT&CK Mapping" [20], the MITRE ATT&CK framework describes the adversarial TTPs (see Section 2.2.1) adopting a structure of four increasingly granular levels: **tactics**, **techniques**, **sub-techniques**, and **procedures**. These elements are organised within one of the three MITRE ATT&CK Matrices described in Section 2.5.1, which visually map out the relationships between them. A matrix is structured as a table where each column represents a tactic, the "why" of an adversary's actions, defining their high-level goals or objectives, such as gaining credential access to infiltrate a network. These tactics provide insight into what the adversary aims to achieve. Still, the matrix does not enforce a linear sequence, meaning an adversary can use varying tactics at different stages of their attack without needing all of them to meet their goal.

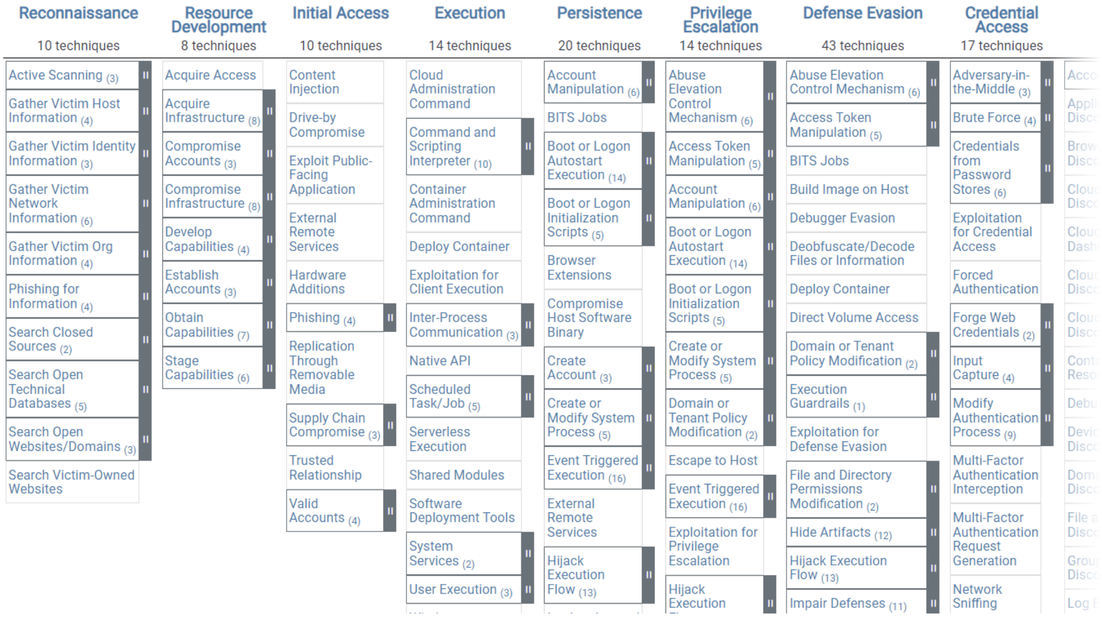
Under each tactic in the matrix, specific techniques are listed (see Figure 2.5), representing the "how" behind achieving a tactic by detailing concrete actions, such as dumping credentials to fulfil the credential access goal. Techniques may apply to multiple tactics and often involve leveraging legitimate system functions for malicious purposes, a concept known as "living off the land". Some techniques are further broken down into sub-techniques, providing more granular, platform-specific descriptions of how a technique might be implemented. For example, dumping credentials

[11] https://attack.mitre.org/

[12] https://attack.mitre.org/matrices/

by accessing LSASS memory [T1003.001] or accessing the "/etc/passwd" and "/etc/shadow" files [T1003.008]. Not all techniques require sub-techniques, but those that do offer additional specificity about the method.

Lastly, procedures describe the "what", the specific actions an adversary took to use a technique or sub-technique, offering real-world examples of attacks. For instance, multiple procedures exist for dumping LSASS memory, each using different tools and methods. They aid in adversary emulation and incident detection, allowing security professionals to understand how attackers have used these techniques in real-world scenarios.



Reconnaissance	Resource Development	Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	
10 techniques	8 techniques	10 techniques	14 techniques	20 techniques	14 techniques	43 techniques	17 techniques	
Active Scanning (3)	Acquire Access	Content Injection	Cloud Administration Command	Account Manipulation (6)	Abuse Elevation Control Mechanism (6)	Abuse Elevation Control Mechanism (6)	Adversary-in-the-Middle (3)	Acco
Gather Victim Host Information (4)	Acquire Infrastructure (8)	Drive-by Compromise	Command and Scripting Interpreter (10)	BITS Jobs	Access Token Manipulation (5)	Access Token Manipulation (5)	Brute Force (4)	Appli Disco
Gather Victim Identity Information (3)	Compromise Accounts (3)	Exploit Public-Facing Application	Container Administration Command	Boot or Logon Autostart Execution (14)	Account Manipulation (6)	BITS Jobs	Credentials from Password Stores (6)	Brow Disco
Gather Victim Network Information (6)	Compromise Infrastructure (8)	External Remote Services	Deploy Container	Boot or Logon Initialization Scripts (5)	Boot or Logon Autostart Execution (14)	Build Image on Host	Exploitation for Credential Access	Clou Disco
Gather Victim Org Information (4)	Develop Capabilities (4)	Hardware Additions	Exploitation for Client Execution	Browser Extensions	Boot or Logon Initialization Scripts (5)	Debugger Evasion	Forced Authentication	Clou Dash
Phishing for Information (4)	Establish Accounts (3)	Phishing (4)	Inter-Process Communication (3)	Compromise Host Software Binary	Create or Modify System Process (5)	Deobfuscate/Decode Files or Information	Forge Web Credentials (2)	Clou Disco
Search Closed Sources (2)	Obtain Capabilities (7)	Replication Through Removable Media	Native API	Create Account (3)	Domain or Tenant Policy Modification (2)	Deploy Container	Input Capture (4)	Clou Disco
Search Open Technical Databases (5)	Stage Capabilities (6)	Supply Chain Compromise (3)	Scheduled Task/Job (5)	Create or Modify System Process (5)	Escape to Host	Direct Volume Access	Modify Authentication Process (9)	Conta Reso
Search Open Websites/Domains (3)		Trusted Relationship	Serverless Execution	Event Triggered Execution (16)	Event Triggered Execution (16)	Domain or Tenant Policy Modification (2)	Multi-Factor Authentication Interception	Debu
Search Victim-Owned Websites		Valid Accounts (4)	Shared Modules	External Remote Services	Exploitation for Privilege Escalation	Execution Guardrails (1)	Multi-Factor Authentication Request Generation	Devic Disco
			Software Deployment Tools	Hijack Execution Flow (13)	Hijack Execution	Exploitation for Defense Evasion	Network Sniffing	Dom Disco
			System Services (2)			File and Directory Permissions Modification (2)		File a Disco
			User Execution (3)			Hide Artifacts (12)		Grou Disco
						Hijack Execution Flow (13)		Log E
						Impair Defenses (11)		

Figure 2.5. MITRE ATT&CK Enterprise Matrix fragment (source: MITRE ATT&CK).

2.5.3 Advantages and Evolutions

The MITRE ATT&CK framework provides numerous advantages for cybersecurity teams, making it a valuable tool for threat detection, incident response, and threat intelligence. First, it offers a common language to describe adversarial behaviour. This standardisation makes it easier for cybersecurity professionals across industries and geographies to collaborate, share information, and enhance collective security. ATT&CK provides a detailed view of how adversaries operate across various environments, from enterprise networks to mobile platforms and industrial control systems, by documenting real-world attack techniques. Moreover, unlike theoretical models, the framework is based on actual observations of threat actors, making it highly relevant to real-world cybersecurity operations. Security teams can prioritise defence mechanisms based on adversaries' most commonly observed tactics, techniques, and procedures (TTPs) and tailor their strategies to specific attack vectors. Given its flexibility, ATT&CK can be adapted for multiple use cases. Security teams can simulate adversary behaviour or integrate the framework into various security platforms, including Security Information and Event Management (SIEM) systems and threat intelligence ones. This opens the doors for automated mapping of events and logs to ATT&CK techniques, making the framework actionable in real-time defence (see Section 2.5.4).

One of the standout features of MITRE ATT&CK is its continuous evolution and improvement. The cybersecurity landscape constantly changes, with new threat actors, techniques, and vulnerabilities emerging regularly. MITRE addresses this dynamic environment by periodically updating its framework to incorporate the latest knowledge on adversary behaviour, including new tactics, techniques, sub-techniques, and procedures. Each version reflects the latest TTPs observed in the field, ensuring that organisations using the framework always work with up-to-date intelligence. In addition, initially focused on enterprise environments, MITRE ATT&CK

has expanded to cover other vital domains, such as mobile and industrial control systems (ICS). This enhancement reflects the growing complexity of the threat landscape and MITRE's commitment to protecting all potential environments. Moreover, the ATT&CK framework thrives on collaboration with the global cybersecurity community. Many of the updates and improvements are informed by insights from security practitioners, vendors, and researchers. This collaborative approach not only keeps ATT&CK relevant but also helps the framework grow in depth and accuracy. Lastly, as new technologies like cloud computing, artificial intelligence, and IoT become integral to enterprise operations, MITRE ATT&CK continuously adapts to include tactics and techniques relevant to these areas. For example, as AI-generated content becomes more prevalent, future iterations of ATT&CK may address how generative AI could be used by adversaries or how defenders can leverage AI to detect complex threats.

2.5.4 Automatic Mapping

One of the challenges cybersecurity professionals face is the time-consuming and error-prone process of manually mapping cyber threat intelligence (CTI) to the MITRE ATT&CK framework. Modern organisations receive vast amounts of cyber threat intelligence from various sources, including reports, alerts, and logs. Nonetheless, CTI reports are often written in natural language, with varying levels of detail and specificity. Security analysts must interpret this unstructured information and align it with the appropriate ATT&CK techniques, which can be subjective and inconsistent across different analysts. Moreover, given the vast number of techniques and sub-techniques in the MITRE ATT&CK matrix, manually mapping every relevant action or indicator can take considerable time. Organisations may lack sufficient resources or expertise to continuously perform this task at scale, particularly when faced with frequent updates to ATT&CK.

This is where automatic mapping tools leveraging natural language processing (NLP) and machine learning come to hand. These technologies can quickly analyse unstructured CTI reports and map relevant behaviours to MITRE ATT&CK techniques. Instead of spending hours or even days manually analysing reports, security teams can leverage automation to complete the task in a fraction of the time. As organisations receive more CTI data, the ability to automatically map it becomes essential. Automation allows processing large datasets in real-time, ensuring no critical intelligence is overlooked. Automated systems also apply consistent logic and criteria when mapping CTI to MITRE ATT&CK. This reduces the subjectivity and variability of manual associations, leading to more reliable and repeatable results. NLP-based tools can analyse large quantities of unstructured data, extracting relevant information and matching it to the appropriate ATT&CK techniques with greater precision. As these tools learn and improve, they can even identify new strategies that may not be immediately apparent to human analysts.

An excellent example of the potential for automatic mapping comes from the work of dessertlab[13], which presents a dataset for automatically mapping unstructured cyber threat intelligence to the MITRE ATT&CK framework using natural language processing. Their project uses advanced NLP techniques to analyse threat reports and map the relevant tactics and methods to the corresponding elements in the framework matrix [21]. This integration of NLP and ATT&CK mapping is a robust advancement in threat intelligence processing, highlighting the growing role of artificial intelligence and machine learning in cybersecurity operations. As more organisations adopt these technologies, the threat analysis and response process will become faster, more efficient, and more aligned with adversary behaviours described in real-time intelligence reports.

2.6 AI Applications in Cybersecurity

Artificial intelligence (AI) has become an essential tool in modern cybersecurity strategies because it processes vast amounts of data, detects patterns, and makes decisions faster than traditional

[13] https://github.com/dessertlab/cti-to-mitre-with-nlp

methods. The growing complexity and sophistication of cyberattacks, including the rise of advanced persistent threats (APTs) (see Section 2.1.3) and zero-day vulnerabilities, have made it clear that conventional security tools are no longer sufficient. They are reactive by nature, often only identifying threats after they have been detected and catalogued. In contrast, AI-based systems offer a more proactive defence mechanism. They can learn from data, adapt to new threats, and improve over time without relying solely on human intervention. This adaptive nature is crucial in an environment where attackers constantly innovate and evolve their techniques. As organizations accumulate vast amounts of data, AI's ability to process and analyze it quickly becomes critical to identifying threats, mitigating risks, and securing systems more effectively.

AI for Threat Detection and Response

One of the key areas where AI has proven impactful is threat detection and response. Traditional security systems often struggle to detect new or subtle threats because they rely on predefined signatures and rule sets. AI systems, by contrast, can analyze large datasets and recognize patterns of behaviour that may signal the presence of malicious activity. These systems can continuously improve their detection capabilities through machine learning algorithms, identifying new threats even if they have never been encountered before. AI-powered detection often relies on anomaly detection, where systems monitor baseline network behaviour and flag unusual deviations. For instance, an AI system might detect abnormal data flows or unusual login times, indicating a possible breach. These systems are precious in large networks where human analysts would struggle to identify threats in real-time [22]. In addition to detection, AI is increasingly being used to automate responses to specific threats. For example, suppose a system detects suspicious activity, such as a potential ransomware attack. In that case, it can immediately isolate the affected systems, prevent data exfiltration, or block the malicious IP addresses. This rapid response can significantly limit the damage caused by cyberattacks, reducing the time attackers have to exploit vulnerabilities.

AI in Malware Detection

Malware, one of the most prevalent and evolving cybersecurity threats (see Section 2.1.3), has also been impacted by AI. Traditional malware detection systems, such as antivirus software, depend on signatures that are digital fingerprints of known malware. However, these signature-based systems fall short as cybercriminals create new malware variants, exceptionally polymorphic malware that changes its code to evade detection. AI changes the game by enabling systems to detect malware based on behaviour rather than signatures. By analyzing how a program behaves in a system, AI can identify suspicious activities associated with malware, such as unauthorized file access or unexpected system modifications, even if it is entirely new. This allows for efficiently detecting zero-day threats [23].

AI for Phishing Prevention

Phishing, where attackers impersonate legitimate entities to trick users into revealing sensitive information, has long been a significant problem in cybersecurity. While traditional email filters can block relevant phishing attempts, more sophisticated attacks, particularly those involving social engineering, can evade these defences. AI is helping to bridge this gap by employing advanced techniques to detect and block phishing attempts [22]. AI systems use natural language processing (NLP) (see Section 2.4.4) to analyze the content of emails and identify phishing attacks, even when the attacker uses carefully crafted language to appear legitimate. This analysis goes beyond simple keyword detection and involves understanding the context and intent behind the communication. Furthermore, as AI systems learn from new phishing attempts, they evolve and become more effective at identifying novel tactics attackers employ. For example, Google's use of AI in Gmail has significantly reduced the number of phishing emails reaching users, with over 99.9% accuracy in filtering these threats [24].

AI in Vulnerability Management

Another crucial application of AI in cybersecurity is vulnerability management. Cybercriminals constantly search for weaknesses in software, networks, or systems they can exploit. Identifying and addressing these vulnerabilities before they are exploited is a race against time, one that AI can help win. AI can analyze historical data and detect patterns that suggest potential breaches, even those that have not yet been discovered. Predictive analysis, powered by machine learning, helps organizations prioritize vulnerabilities by predicting the likelihood of exploiting them. This ensures that the most critical ones are addressed first. In addition, AI-driven tools can automate the process of patch management, identifying vulnerable systems and deploying necessary updates before attackers can take advantage of the weaknesses.

AI is also being employed in bug detection. Scanning codebases can identify potential security flaws and suggest fixes, greatly enhancing the security of software development processes. This is particularly important for organizations developing their software or using third-party code, where hidden vulnerabilities can introduce serious risks.

Future of AI in Cybersecurity

Looking ahead, the role of AI in cybersecurity is likely to expand as both cyberattacks and defence mechanisms grow more sophisticated. One promising development is the rise of autonomous defence systems, where AI could operate with minimal human intervention to detect, respond to, and even prevent cyberattacks. These systems could continuously learn from new data, adapt to changing threats, and act decisively in real-time. As AI algorithms continue to improve, we may see more advanced applications, such as AI being used to predict the methods attackers will use, enabling organizations to stay ahead of the curve. Rather than replacing human analysts, AI will increasingly augment their abilities, allowing them to focus on more strategic tasks while it handles the more routine aspects.

This growing role of AI in cybersecurity also intersects with the advancements in honeypot technology, where AI is being increasingly utilized to enhance the efficiency and effectiveness of honeypot systems. Integrating AI with honeypots marks a significant development in cybersecurity, forming the basis of cutting-edge approaches to deception-based security strategies.

2.6.1 Honeypot and AI: State of the Art

Honeypots have long been a crucial tool in cybersecurity, functioning as decoy systems to lure attackers and analyze their behaviour (see Section 2.2). Traditionally, they were static systems designed to attract specific attacks or gather intelligence on malicious actors' tactics, techniques, and procedures (TTPs). However, with the increasing complexity and scale of cyberattacks, there is a growing need for more dynamic and intelligent honeypot systems. AI and machine learning are crucial in advancing honeypot technology and adapting it to modern security challenges. Integrating them with honeypots represents a significant leap forward, enabling more dynamic, intelligent, and responsive systems to detect better and analyse cyber threats.

LLM in the Shell

A cutting-edge example of AI-enhanced honeypot technology is the shelLM project [25], developed by the Stratosphere Laboratory at the Czech Technical University in Prague. The shelLM project focuses on creating a honeypot that mimics natural system shells, providing an interactive environment for attackers while collecting valuable information about their methods [26]. Unlike traditional honeypots, which experienced attackers may more quickly detect, shelLM leverages generative AI to continuously adapt its behaviour, making it indistinguishable from legitimate systems. This adaptability not only enhances the realism of the honeypot but also ensures that attackers remain engaged, allowing the system to gather more detailed intelligence about their behaviour. The dynamic nature of shelLM will enable it to react in real-time to attackers' actions, making the honeypot more effective at trapping sophisticated cybercriminals. The shelLM project exemplifies the power of combining honeypot technology with AI to enhance cybersecurity, offering a more responsive, adaptive, and effective defence mechanism against increasingly complex network threats.

Industry 4.0

One notable real-world application of AI in honeypots is presented by Lee, Abdullah, Jhanjhi, and Kok (2021) in their study titled "Classification of botnet attacks in IoT smart factory using honeypot combined with machine learning" [27]. Their work demonstrates the effectiveness of combining honeypots with machine learning to classify and analyze botnet attacks within an Internet of Things (IoT) environment, specifically a smart factory. The researchers deployed honeypots to simulate vulnerable IoT devices, allowing attackers to interact with them. AI-based machine learning algorithms were then used to analyze the traffic data generated by these interactions, resulting in accurate classification of botnet attacks. This study highlights how the integration of AI can significantly enhance honeypot capabilities in complex, connected environments like IoT-based smart factories. AI's ability to process large amounts of data in real-time enables these honeypots to detect known attack patterns and identify novel or evolving threats. The machine learning models used in the study allowed the system to adapt dynamically, adjusting to new forms of botnet activity and improving the accuracy of threat detection over time.

Adaptive Honeypot Deployment

In their paper titled "An Adaptive Honeypot Configuration, Deployment and Maintenance Strategy", Daniel Fraunholz, Marc Zimmermann, and Hans D. Schotten propose a solution to a long-standing challenge in honeypot technology: the complex and resource-intensive nature of manual deployment and maintenance. Traditional honeypots require configuration and constant oversight from network administrators, making them difficult to scale and maintain in modern cybersecurity environments. Their method eliminates manual configuration by featuring an automated identification mechanism for devices and machines within a network. These entities are analyzed and clustered based on their characteristics, allowing the honeypot to be intelligently deployed where it can be most effective [28]. This adaptive method ensures the honeypot remains relevant and functional without ongoing maintenance, significantly improving traditional, manually configured honeypot systems.

2.6.2 Comparison with Traditional Honeypots

Traditional honeypots have played a crucial role in cybersecurity for years. However, they have several limitations (see Table 2.4). First, their configuration is mainly manual, requiring significant time and effort from network administrators. Each honeypot must be tailored to the network's specific conditions, making the process prone to human error. Furthermore, once deployed, traditional honeypots need constant maintenance to stay relevant as attack strategies evolve and change over time. In contrast, AI-integrated honeypots bring automation to configuration and maintenance processes. AI techniques, especially machine learning, allow honeypots to adjust based on real-time data dynamically, eliminating manual intervention. AI models analyze network traffic, detect changes in threat landscapes, and reconfigure honeypots to optimize their deployment and effectiveness. This reduces the administrative burden and ensures the honeypot remains functional and relevant without frequent updates. Regarding deployment, traditional honeypots rely on the network administrator's knowledge and experience. They are typically deployed in fixed environments, where attackers are expected to target specific vulnerabilities. This static nature limits their flexibility and ability to adapt to cyber threats. On the other hand, AI-integrated honeypots are far more scalable and flexible. They can analyze the network and cluster machines, intelligently deploying honeypots in areas where they will be most effective.

Traditional honeypots are also limited in their ability to respond to emerging threats. These systems often rely on predefined signatures or behaviours to detect attacks, making them less effective against zero-day or advanced persistent threats (APTs). However, AI-enhanced honeypots can learn from new data and recognise anomalous behaviours, enabling them to detect and mitigate novel attack vectors in real-time. Another key difference lies in data analysis. Traditional honeypots passively log interactions with attackers, requiring manual analysis by cybersecurity teams, which can be time-consuming. In contrast, AI-integrated honeypots automatically analyze the data collected during attacks, using machine learning algorithms to detect patterns, classify threats, and provide real-time insights. This speeds up the response to attacks and reduces the likelihood of human error during analysis.

Moreover, AI-integrated honeypots are more difficult for attackers to detect. Skilled malicious actors often identify traditional ones due to their static and predictable nature. AI-based continuously adapt and simulate natural systems, making it much harder to distinguish them from legitimate targets. As a result, these AI-driven systems can engage attackers for extended periods, capturing more valuable data on their tactics, techniques, and procedures (TTPs). In terms of cost and resource efficiency, traditional honeypots are expensive in the long run due to the high human resource costs for configuration, monitoring, and maintenance. Although AI-enhanced honeypots may require a significant initial investment, particularly in AI model development, they offer substantial long-term savings by automating many processes and reducing the need for constant human oversight.

Feature	**Traditional Honeypots**	**AI-Integrated Honeypots**
Configuration	Manual, static, and time-consuming.	Automated, dynamic, and self-adjusting using machine learning.
Maintenance	Requires constant manual updates.	Automated maintenance with minimal human intervention.
Deployment Strategy	Fixed, based on administrator knowledge.	Intelligent deployment using real-time data analysis.
Scalability	Limited by manual effort.	Highly scalable due to automation.
Response to Emerging Threats	Limited to known threats with predefined signatures.	Adaptive, capable of detecting new and novel attack patterns.
Attack Detection	Passive logging for post-attack analysis.	Active engagement, real-time detection, and response.
Anomaly Detection	Static detection rules.	AI-driven, based on pattern recognition and behaviour analysis.
Data Analysis	Manual analysis of logs and behaviour.	Automated, real-time data analysis using AI.
Attractiveness to Attackers	Can be detected by experienced attackers.	Harder to detect due to continuous adaptation and realism.
Cost and Resource Efficiency	High resource costs for manual processes.	More cost-effective in the long term due to automation.

Table 2.4. Comparison between Traditional and AI-Integrated Honeypots

2.6.3 Advantages and Limitations of Honeypots integrated with AI

Integrating generative AI into honeypot systems represents a significant advancement in cybersecurity, offering numerous benefits across various dimensions. These AI-enhanced honeypots provide advanced attack detection, real-time analysis, and automation while improving adaptability and scalability. However, as with any technological advancement, they come with certain limitations that must be considered alongside their advantages.

Advanced Attack Detection

One of the most prominent advantages of AI-integrated honeypots is their real-time analysis capabilities. Generative AI can process and analyse vast amounts of data instantaneously, enabling immediate detection of cyberattacks. This rapid response is critical in preventing widespread network damage and reducing attackers' dwell time. Moreover, generative AI excels in anomaly detection, identifying unusual patterns of behaviour that traditional signature-based detection methods might miss. AI algorithms can detect subtle deviations in system behaviour or communication traffic, allowing for the identification of novel attack vectors or previously unknown threats. While AI's real-time capabilities are a significant strength, detection accuracy relies heavily on the quality and diversity of the training data. If an AI system has not been adequately trained on a wide range of attack behaviours, there is a risk that new types of attacks could still go unnoticed.

Automation and Efficiency

Generative AI can dramatically enhance data collection and analysis automation, significantly reducing the need for human intervention. This frees cybersecurity professionals to focus on higher-level strategy and decision-making rather than routine tasks. Another key benefit is the reduction of false positives. By leveraging advanced machine learning techniques, AI-integrated honeypots can more accurately distinguish between legitimate activity and malicious behaviour. This reduces the volume of false alarms and allows security teams to prioritise actual threats, leading to more efficient threat management. Despite the improved accuracy, AI systems are not infallible. In some instances, false positives may still occur, particularly in environments with unusual but legitimate activity that AI algorithms may misclassify as a threat. Furthermore, while AI reduces manual workload, it may introduce the need for highly skilled personnel to manage and interpret AI-generated insights, which can be a resource challenge for smaller organisations.

Adaptability and Continuous Learning

One of the critical advantages of AI-driven honeypots is their continuous learning capability. Machine learning algorithms improve over time as they process more data from cyberattacks, making them increasingly proficient at detecting and responding to threats. Unlike traditional systems, which require manual updates to remain effective, AI honeypots can adapt to new forms of cyberattacks, keeping network security up to date without significant manual input. AI also provides superior adaptability to new threats. As attackers develop novel techniques, AI-powered honeypots can quickly learn and adjust, maintaining a robust security posture despite rapidly evolving cyber threats. While AI offers the ability to learn and adapt, its effectiveness depends on the availability of sufficient, high-quality data to train the system. If AI models are not exposed to various attack vectors during the learning phase, they may struggle to adapt effectively to new or highly sophisticated threats. Additionally, adversarial attacks can be launched to exploit vulnerabilities in AI models, misleading the system and compromising its defensive capabilities.

Detailed Attack Analysis

Generative AI also enables detailed profiling of attackers using the data collected during honeypot interactions. By analysing methods and tactics, AI can help security teams build comprehensive profiles of threat actors, including their preferred techniques and objectives. This detailed insight supports the creation of more effective countermeasures and defensive strategies. Another significant benefit is the ability of AI to map tactics, techniques, and procedures (TTPs) to frameworks like the MITRE ATT&CK matrix (see Section 2.5). This helps organisations better understand attack patterns and improve their preparedness for similar threats in the future. While AI can provide detailed analysis, there is the risk of over-reliance on automated insights. Human oversight remains crucial to interpret the data effectively and apply it strategically. In some cases, AI-generated profiles might be too generic or incomplete without human verification, potentially leading to gaps in security coverage.

Improved Network Security

AI can also enhance proactive protection. Through predictive analysis, AI can anticipate potential attack scenarios based on patterns observed in the data. This allows organisations to adopt preventive measures, such as patching vulnerabilities or adjusting firewall settings, before an attack occurs. Furthermore, AI's ability to integrate with security systems such as Security Information and Event Management (SIEM) or Intrusion Detection Systems (IDS) provides a layered defence mechanism. While AI offers advanced protection, it can sometimes lead to false confidence. Organisations might assume that AI-powered systems are infallible, reducing manual vigilance. Moreover, integration with existing systems requires careful configuration to ensure seamless operation, and misconfigurations can introduce new vulnerabilities.

Scalability

Generative AI significantly improves the scalability of honeypot systems. Artificial intelligence enables honeypots to handle and analyse vast amounts of data, making them suitable for organisations of any size. This makes AI-powered honeypots particularly effective in large, distributed environments such as cloud infrastructures and IoT ecosystems. Moreover, the ease of implementation and maintenance offered by AI reduces the workload on IT teams. Although scalability is a clear advantage, the initial setup costs and the need for advanced AI expertise can be prohibitive for smaller organisations or those with limited resources. Furthermore, AI-powered honeypots may require significant computational power as the system scales, increasing infrastructure costs.

Support for Research and Development

AI-enhanced honeypots can collect large amounts of data on cyberattacks, which is invaluable for cybersecurity research. This data can be used to develop new defensive techniques, improve existing security tools, and foster innovation in the field. The continuous learning capabilities of AI also contribute to ongoing innovation, as the system constantly enhances its understanding of the threat landscape. While the data collected is valuable, it may also introduce privacy concerns if sensitive information is inadvertently captured and not adequately anonymised (see Section 2.2.5). Additionally, organisations may face challenges in securely storing and managing the large datasets required for AI training and research.

Reduced Operational Costs

The operational efficiency brought by automation helps reduce the costs associated with manual system management. AI can detect and respond to threats more quickly, reducing downtime and minimising the financial impact of successful attacks. Automating data analysis and threat detection also decreases the need for a large security team, providing further savings. While AI-driven systems may reduce long-term operational costs, the upfront investment in AI infrastructure and expertise can be significant. As already said, smaller organisations may struggle with these initial costs, which can offset the savings realised from reduced manual intervention.

Chapter 3

Methodology

Building on the foundational concepts discussed in the literature review, this chapter outlines the methodology around the SYNAPSE project. It integrates honeypot technologies with generative AI, creating a framework for studying and enhancing honeypot effectiveness in cybersecurity. The SYNAPSE-to-MITRE extension adds more sophistication by incorporating machine learning models and aligning threat detections with the MITRE ATT&CK framework. This provides a structured and standard-aligned approach to identifying and responding to attacks. The chapter details each project component (see Figure 3.1 for a complete overview), starting with the design and implementation of SYNAPSE. It also examines the SYNAPSE-to-MITRE extension and concludes with an analysis of the case studies conducted to assess the project's performance. Each phase is systematically described, ensuring transparency and rigour in the methods used and offering a clear path for replication in future research.

3.1 SYNAPSE

SYNAPSE[1], or **Synthetic AI Pot for Security Enhancement**, is an open-source advanced honeypot system entirely written in Python, designed to enhance digital defences by leveraging artificial intelligence. The name "SYNAPSE" draws a parallel to the neural connections in the human brain, emphasising the system's role in quickly processing, analysing, and responding to cyber threats like how synapses transmit signals in a neural network. The term "Synthetic AI Pot" emphasises using artificial intelligence to enhance the traditional honeypot framework. Unlike conventional ones, which are often passive and static, SYNAPSE adapts dynamically, using AI to learn from intrusions and anticipate evolving attack strategies. By mimicking natural systems and continuously adjusting their defences based on attacker behaviour, SYNAPSE becomes a powerful tool for understanding the latest threats and enhancing overall security.

In a modern cybersecurity environment, where threats grow more complex, SYNAPSE not only traps attackers but also provides real-time intelligence to fortify security systems, turning what could be a vulnerable point into an active defence mechanism, much like how human synapses learn and adapt over time (see Figure 3.2 for an overview).

3.1.1 Classification

SYNAPSE can be classified as a **low-interaction** honeypot because it primarily simulates services like an SSH server and a MySQL server, focusing on gathering initial attack attempts and patterns. However, one could argue that it leans toward a higher-interaction honeypot since it mimics a complete Linux operating system, including a functioning filesystem. This level of simulation

[1] https://github.com/eneagizzarelli/SYNAPSE

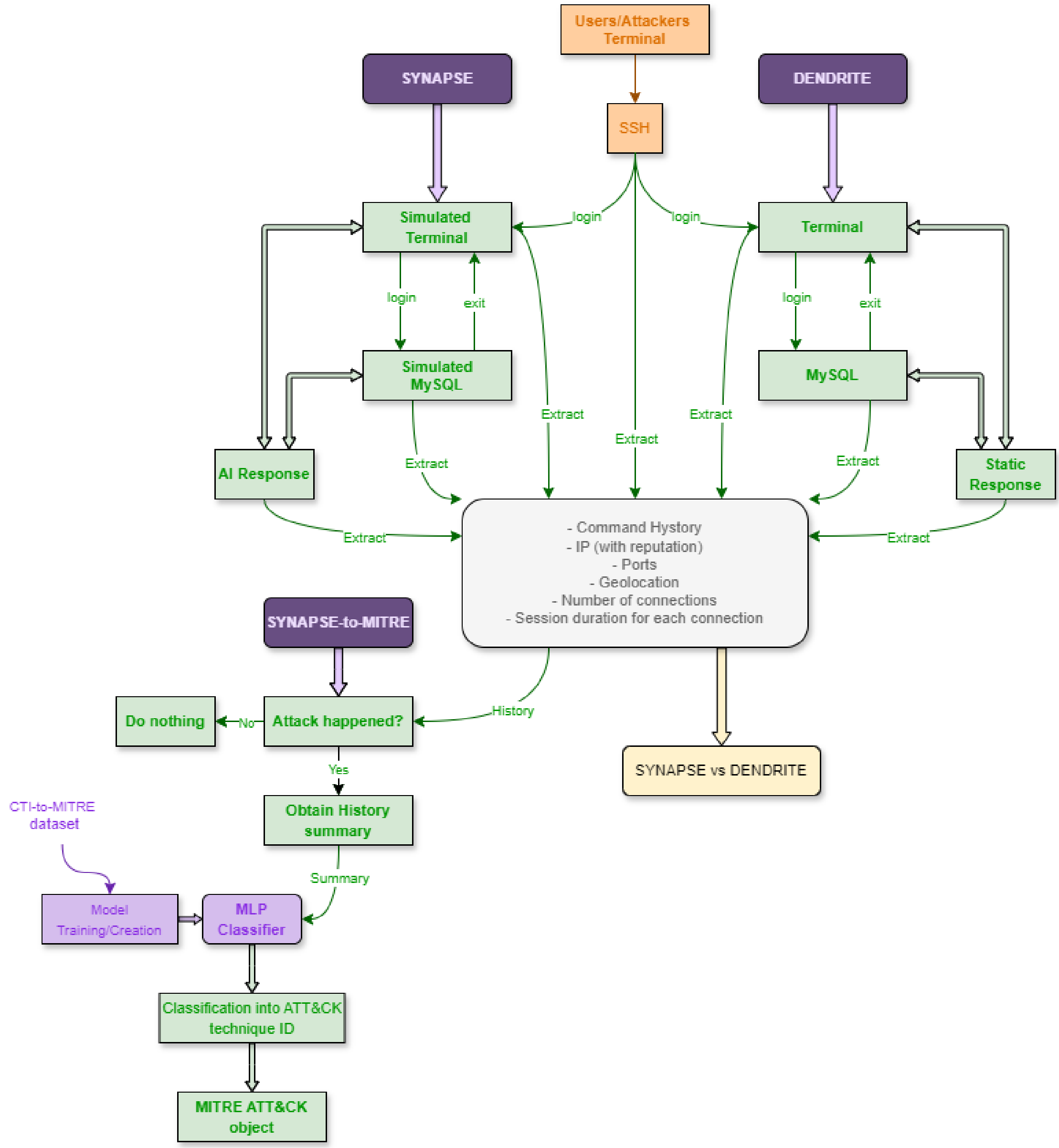


Figure 3.1. Honeypot and Generative AI Project Overview.

allows attackers to interact with the system in more realistic ways, giving the illusion of deeper engagement while still maintaining the security and control typical of low-interaction systems.

From the perspective of a **server** honeypot, SYNAPSE is explicitly designed to simulate server environments, which makes it effective at attracting attacks targeting typical server vulnerabilities. Focusing on critical services like SSH and MySQL is a reliable decoy for attackers looking to exploit server-side weaknesses.

As a **research** honeypot, SYNAPSE shines by providing valuable data for analysing attack patterns and methodologies. Its AI-driven adaptability enhances its research capabilities by learning from interactions, evolving with new threats, and offering insights for strengthening broader cybersecurity measures.

Lastly, SYNAPSE operates as a **virtual** honeypot, running in virtualised environments rather than physical hardware. This design choice provides flexibility and scalability, allowing multiple instances to be deployed without needing extensive physical resources and enabling it to monitor a wide range of attack vectors while remaining cost-efficient.

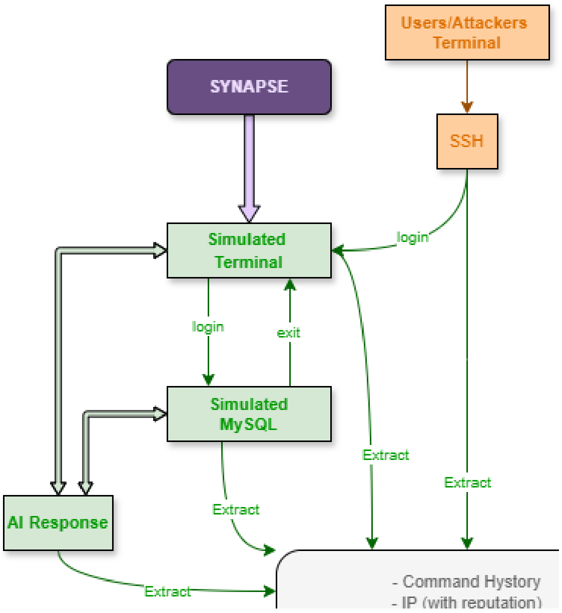


Figure 3.2. SYNAPSE Overview.

3.1.2 Placement and Working Environment

SYNAPSE is a Python-based code hosted within a virtual machine (VM) on Oracle Cloud Infrastructure (OCI)[2]. Initially, there were efforts to deploy SYNAPSE inside an Amazon AWS EC2[3] VM, but several factors led to the decision to shift to OCI. One of the critical advantages of choosing it was the higher resource allocation available in the free tier, particularly in terms of memory. Oracle Cloud offers 24 GB of RAM within its free plan, compared to AWS's 1 GB. Given the resource demands of running generative AI-based processes for SYNAPSE, the more significant RAM in OCI made it more suitable for handling complex simulations without performance degradation.

The current deployment of SYNAPSE is hosted on an OCI VM based in Frankfurt, Germany. Placing the VM in Europe was a strategic decision, as the server's location can affect network latency and legal considerations. This could provide lower latency for users or attackers operating within the region, enhancing the honeypot's realism. Additionally, placing the VM in Europe rather than America may offer different data privacy and cybersecurity regulations that could be advantageous depending on the target audience or nature of the research being conducted. SYNAPSE is generally deployed in a cloud environment with a public IP address, making it accessible from external networks. This configuration attracts potential attackers who can interact with the honeypot in real-time. By leveraging the flexibility and scalability of cloud infrastructure like OCI, SYNAPSE can dynamically adjust to varying traffic loads and provide a reliable environment for collecting valuable data from malicious activities, all while optimising resource use through OCI's robust free tier.

In more detail, when a user or attacker connects to the SYNAPSE honeypot via SSH, they must log in using the specific username "enea" and a password, which has been deliberately set to "password" to make it easy to discover. This approach is designed to encourage attackers to attempt logging into the system. To ensure that the SYNAPSE script is executed upon login, modifications have been made to the */etc/ssh/sshd_config* file. Specifically, the configuration includes the following directive:

[2] https://www.oracle.com/cloud/

[3] https://aws.amazon.com/it/ec2/

```
Match User enea
    ForceCommand /home/enea/SYNAPSE/scripts/startSYNAPSE.sh
```

This command ensures that when the user "enea" logs in, the SYNAPSE Python script is automatically executed through the "ForceCommand" directive. As a result, the attacker is enclosed within the SYNAPSE simulation immediately after authentication. This environment, supported by generative AI, mimics an actual Linux terminal but is, in reality, a wholly controlled honeypot system. The attacker cannot escape the SYNAPSE script to access the exact system or files. If the program is terminated or exited, the SSH session is automatically closed, cutting off the attacker's connection to the VM. This configuration allows seamless and controlled interaction while capturing valuable information about the attacker's behaviour within the simulated environment.

LLM

The current version of SYNAPSE uses the "gpt-4o" model, which has proven to be the most efficient and capable for the specific requirements of this project. Throughout development, various models have been tested and evaluated, each offering unique advantages and challenges. Initially, "gpt-3.5-turbo-0125" was used, followed by "gpt-4-0125". These models significantly improved language understanding, response generation, and overall coherence. However, they still had certain limitations, mainly when simulating complex technical environments like a Linux terminal. In recent months, the landscape of language models has evolved rapidly, leading to the deployment of "gpt-4o", representing a significant performance leap. This model has been specifically optimised for tasks that require complex contextual understanding, including the realistic simulation of environments like a Linux terminal. Its ability to interpret commands and generate coherent, contextually accurate responses made it ideal for SYNAPSE. Compared to earlier versions, "gpt-4o" offers enhanced accuracy in interpreting system-level commands, mimicking the behaviour of an actual OS with greater precision and consistency. This is crucial in maintaining the realism of the honeypot and keeping attackers engaged in the simulated environment.

During the initial phases of SYNAPSE development, "Google Gemini" was also evaluated as a potential language model. However, its performance in replicating a Linux OS terminal was notably less effective than the GPT models. While Google Gemini is an advanced model with strengths in general natural language processing, it struggled with the technical intricacies required to mimic system interactions and command outputs and maintain a convincing terminal environment. The GPT models, particularly "gpt-4o", have been better suited for this task due to their superior ability to manage and respond to complex, structured inputs, essential for simulating realistic interactions in a honeypot.

3.1.3 Genesis: from shelLM to SYNAPSE

The development of SYNAPSE began by building on the "shelLM" project [25], a decision rooted in both efficiency and practicality. shelLM has already provided a robust framework for key honeypot functionalities integrated with the flexibility of generative AI [26]. These foundational features were well-established and had been proven effective in shelLM, making it unnecessary to "reinvent the wheel". Using it as a starting point handed the time and resource drain that would have come with redeveloping these core components from scratch. This approach shifted the focus toward enhancing the project with more advanced capabilities, allowing it to build a more innovative and evolved solution while leveraging the strengths of an already-tested system.

3.1.4 Simulated Terminal

The core functioning of SYNAPSE remains closely aligned with shelLM's. It simulates a Linux OS terminal, a dynamic fake file system, and command responses [25] by leveraging generative AI, particularly Large Language Models (LLMs).

When a user or, in the case of a bad actor, an attacker obtains the IP address of the machine hosting the SYNAPSE honeypot, the flow of operations begins once they attempt to connect via

SSH. If the connection is successfully established, the SYNAPSE system is triggered, and the attacker is seamlessly enclosed in a simulated, fake environment (see Section 3.1.2). At this point, the user believes they have accessed a legitimate Linux operating system, as the interface closely mimics the appearance and behaviour of an actual terminal. However, behind the scenes, none of the interactions reflect an actual system. Instead, SYNAPSE leverages generative AI to simulate the entire environment. Every piece of information the user sees, from the filesystem structure to directory listings and terminal outputs, is generated in real-time by the AI. The user believes they are interacting with a real Linux machine, but nothing physically exists on the device; it is just a Python program inside which they are constrained.

Interaction Flow

Once the connection is established, SYNAPSE captures the user's IP address and inserts it into the typical "last login" message displayed when connecting to a Linux machine via SSH, reinforcing the illusion of a legitimate system. The simulated terminal prints the message, and the user is prompted to input a command or perform further actions.

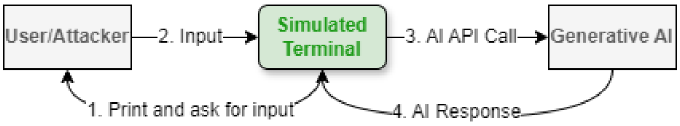


Figure 3.3. Simulated Terminal Interaction Flow.

As the user inputs a command into the terminal, SYNAPSE's generative AI interprets and generates coherent and realistic responses within the simulated environment (see Section 3.1.5 to understand how AI's requests and responses are processed). Each command, whether navigating directories, accessing files, or performing administrative tasks, triggers the AI to generate appropriate output that aligns with typical Linux OS behaviour. This process continues throughout the user's interaction, with SYNAPSE maintaining the illusion of a functioning system while capturing valuable data about the user's activities.

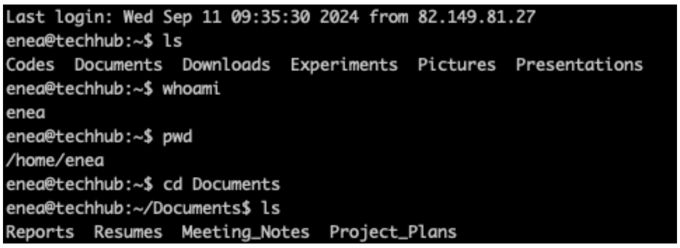
Prompt

SYNAPSE's simulated terminal's ability to function effectively relies on a specific type of prompt engineering known as "Persona Prompting", mixed with other techniques like "Few-Shot Prompting" and "Chain-of-Thought Prompting". The first involves instructing the generative AI to adopt the persona of a Linux OS terminal, ensuring that it behaves and responds in a manner consistent with how an actual Linux system would. A carefully crafted prompt shapes The AI's behaviour, setting the stage for its role. For example, the prompt begins with:

You are a Linux OS terminal. You act and respond exactly as a Linux terminal. You will respond to all commands just as a Linux terminal would . . .

This explicit instruction guides the AI, enabling it to generate accurate and realistic responses to any command input by the user. Defining the AI's "persona" as a Linux terminal can simulate the operating system's behaviour, from responding to basic navigation commands to executing more complex tasks. This approach ensures that the user remains convinced they are interacting with a natural system while the AI seamlessly adapts to various inputs in real-time.

Building on the original prompt developed by the actual creators of shelLM, this work reordered, refined and adapted it to suit a new AI model, focusing on enhancing clarity and realism. These modifications were essential to ensure that the AI's responses more accurately mimic the behaviour of an actual Linux OS terminal. For example, one significant refinement involves instructing the AI to hide files starting with a dot (such as ".gitignore") when executing a basic *ls* command, replicating how a genuine Linux system would handle such files. Additionally, the prompt was enhanced to dynamically reflect the username used during the SSH login and the correct IP address in the “last login” message. At the same time, the AI dynamically generates the hostname. These adjustments contribute to a more immersive and believable experience, ensuring that the AI-generated environment closely resembles an actual terminal, reinforcing the effectiveness of the SYNAPSE honeypot (see Figure 3.4).



```
Last login: Wed Sep 11 09:35:30 2024 from 82.149.81.27
enea@techhub:~$ ls
Codes  Documents  Downloads  Experiments  Pictures  Presentations
enea@techhub:~$ whoami
enea
enea@techhub:~$ pwd
/home/enea
enea@techhub:~$ cd Documents
enea@techhub:~/Documents$ ls
Reports  Resumes  Meeting_Notes  Project_Plans
```

Figure 3.4. Simulated Terminal after successful SSH login.

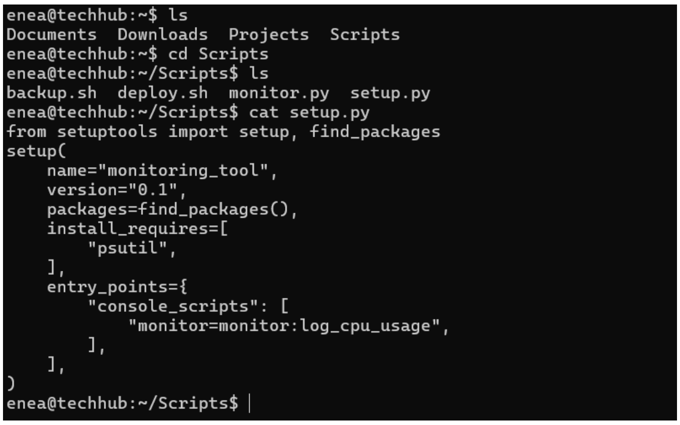
Features

SYNAPSE is designed to closely replicate a realistic Linux terminal environment, providing an immersive experience for users or attackers while maintaining complete control of the system’s behaviour. Several key features ensure the simulation remains convincing while maintaining security protocols and capturing valuable interaction data.

One of the primary features of SYNAPSE is its restriction on using *sudo*. In a natural Linux system, *sudo* allows users to execute commands with elevated privileges. However, in SYNAPSE, any attempt by the user or attacker to run a command with *sudo* will immediately close the SSH connection. This simulates a security mechanism where such privileged access is not permitted in this environment, discouraging further intrusion and preventing users from attempting to escalate their privileges. The connection closure provides a subtle yet decisive way to protect the honeypot and terminate potential misuse. Another feature of SYNAPSE is its ability to provide realistic responses to incorrect or invalid commands. Just as an actual Linux terminal would return an error message when an unrecognised or improperly formatted command is entered, SYNAPSE’s generative AI generates appropriate error responses. This ensures that users attempting to test the system with erroneous inputs will receive plausible feedback, further reinforcing the authenticity of the environment. Responses such as “command not found” or specific syntax-related errors mimic actual terminal behaviour, helping maintain the illusion of a fully functioning Linux system.

Moreover, SYNAPSE does not rely on a pre-existing or real filesystem; instead, it is dynamically generated by the AI. The latter creates realistic file and directory structures whenever the user interacts with the system, navigating directories, listing contents, or accessing documents. This includes generating names for files and directories, defining their locations within the hierarchy, and populating contents to correspond to the typical files found in a Linux system, specifically an IT one, as written in the prompt. Because there is no underlying, static filesystem, the AI creates a new version of the file structure every time a relevant command is executed. For instance, when the user runs the *ls* command in a given directory, SYNAPSE’s AI generates a plausible list of files and directories that typically reside there. Upon issuing a *cat* command, the content of files is generated dynamically, ensuring the output appears consistent with what would be expected from an actual file in the system. This content might include standard configurations, logs, or text files created on the fly to simulate actual data. Each interaction is crafted to match real-world scenarios, further deepening the deception and engagement of attackers (see Figure 3.5).

Several enhancements to the shelLM code have been introduced in SYNAPSE, further improving the simulation. While the AI cannot automatically handle terminal screen clearing through commands like *clear*, this functionality has been enforced through a Python system call. This ensures that users can clear the terminal just as they would in a natural Linux environment, maintaining the illusion. SYNAPSE also supports command history navigation using the up and down arrow keys, allowing users to recall previously entered commands. Another significant enhancement is the ability to handle all possible ways a user might try to exit the terminal. The



```
enea@techhub:~$ ls
Documents  Downloads  Projects  Scripts
enea@techhub:~$ cd Scripts
enea@techhub:~/Scripts$ ls
backup.sh  deploy.sh  monitor.py  setup.py
enea@techhub:~/Scripts$ cat setup.py
from setuptools import setup, find_packages
setup(
    name="monitoring_tool",
    version="0.1",
    packages=find_packages(),
    install_requires=[
        "psutil",
    ],
    entry_points={
        "console_scripts": [
            "monitor=monitor:log_cpu_usage",
        ],
    ],
)
enea@techhub:~/Scripts$ |
```

Figure 3.5. Dynamic Filesystem Structure and Content Generation.

system appropriately responds to *exit*, *logout*, or when the user presses *CTRL+C* or *CTRL+D*. SYNAPSE generates the correct message in each case, such as "logout" before closing the SSH connection. This ensures that users encounter the expected behaviour when trying to exit while maintaining control over the session termination. Lastly, an attempt was made to implement tab auto-completion by leveraging AI to generate completions of partially entered commands or file names. While this feature showed some success, it did not consistently replicate the accuracy of an actual Linux terminal's auto-completion, but it remains an area for potential future refinement. See Appendix A for more details.

3.1.5 AI Requests and Responses

SYNAPSE interacts with OpenAI APIs to leverage the capabilities of the GPT models, specifically "gpt-4o" in this case, allowing seamless communication between the honeypot code and the model to generate realistic responses. Whenever a user or attacker inputs a command into the SYNAPSE terminal, an API request is made to OpenAI's servers. This API call includes several key components:

- **Model**: the specific model being used (e.g., "gpt-4o").
- **Prompts**: structured input the GPT model processes to generate a response. These prompts are divided into three main types:
  - **System**: defines the overarching behaviour of the model. In the context of SYNAPSE, the AI is instructed to behave as a Linux OS terminal (see Section 3.1.4).
  - **User**: represents the input from the person interacting with the SYNAPSE terminal. For example, when an attacker inputs the command *ls* or *cd /home*, this command is sent as part of the "user" prompt to the GPT model. It tells the LLM what the user is trying to do or ask, and the model then generates a response based on this input.
  - **Assistant**: response the GPT model provides based on the "system" and "user" prompts. This output will be displayed in the SYNAPSE terminal, simulating what an actual Linux terminal would return. For instance, if the attacker issues the *ls* command, the "assistant" prompt would include the AI-generated directory listing as if the terminal were responding in real-time.

Once the API call is made, the GPT model processes the input. Using the "system" prompt frames the overall behaviour (in this case, responding as a Linux terminal). It interprets the "user" prompt, understanding it as a command that needs a terminal-like response. It then generates a response as the "assistant" prompt, which mimics the behaviour of a Linux terminal based on the command provided by the user. The GPT model's ability to understand the context and maintain coherence across multiple interactions is critical for SYNAPSE to simulate an actual terminal convincingly. The API enables this by facilitating the exchange of prompts and responses between SYNAPSE and the OpenAI model. Once the LLM has processed the input and generated a response, the assistant's output is returned to SYNAPSE via the API. This output is displayed in the terminal as if it were the actual response from a real Linux OS. The process repeats with each command inputted by the user, ensuring continuous interaction within the simulated environment.

3.1.6 AI Memory and Multiple Sessions

SYNAPSE leverages AI "memory" by ensuring the system remains context-aware during interactions. Each time the user or attacker issues a new command, the entire terminal prompt, all previously entered commands, and their corresponding responses are sent to the AI again. This method allows the LLM to retain context throughout the session, ensuring consistency in its responses and maintaining a coherent environment. For example, if the *ls* command is used to list files in a directory, subsequent uses of *ls* in the same directory will return the same file list. Similarly, if a file is accessed with the *cat* command, the same content will be shown if the file is reopened. This approach ensures that the system behaves predictably and realistically, reinforcing the illusion of a fully functioning Linux terminal.

However, this method also has limitations, particularly regarding the "context window" of the AI model. In the case of "gpt-4o", the model can process a maximum number of tokens or text data in a single request, specifically 128k. The conversation history grows as the user interacts with SYNAPSE, potentially exceeding the model's context window. When this limit is approached, the AI may lose the ability to recall some earlier parts of the session, which could impact the continuity and consistency of the simulated environment. To mitigate this, SYNAPSE must align the session history within the available context window. This might involve selectively trimming the conversation's older or less relevant parts while preserving critical information. For instance, essential details such as file structures, directories, and previous outputs likely to be revisited by the user can be prioritised. At the same time, minor or redundant interactions may be omitted to keep the AI within its context limits. While this strategy ensures that SYNAPSE operates effectively within the model's constraints, the limitations of the context window do present challenges, particularly in longer sessions where a significant amount of information needs to be retained.

By following the same reasoning, SYNAPSE is designed to handle **multiple user sessions** effectively, ensuring the system remains context-aware for each user or attacker. This capability is made possible by tracking connections based on the IP address. Each time a user connects to the honeypot, the system checks whether it is the first connection from that specific IP address or if it has connected previously. If it is the first connection, a new log file is created to store the session's commands and responses, allowing the AI to begin tracking the user's actions from that point onward. If it has connected before, SYNAPSE loads the existing log file associated with that IP address. This log, containing the user's history of previous interactions, is appended to the terminal prompt and sent to the AI to obtain continuity. The system ensures that each session is contextually aware and consistent by feeding the model with the entire session history, the previous logs and the new commands or responses. This allows SYNAPSE to maintain the appearance of a persistent environment where users can return to previously generated files, directories, and outputs as if they had never left. This process works in cycles: after checking and loading the relevant log file, the AI is provided with the session's context, including previous interactions, and new requests from the user are processed in the same manner. This session management system is crucial for maintaining a seamless and convincing experience. It ensures that users feel like they are working within a stable and ongoing Linux terminal environment, regardless of whether they interact with the honeypot for the first time or return for a subsequent session.

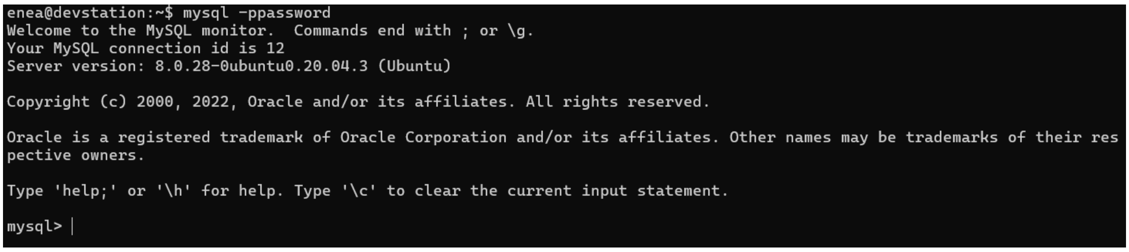
3.1.7 Simulated MySQL

In addition to a Linux terminal environment, SYNAPSE simulates a MySQL service to enhance the realism and increase the honeypot's attractiveness. MySQL is a popular and widely used database management system, making it an appealing target for attackers seeking to exploit vulnerabilities, steal data, or gain further access to a compromised system. By simulating this service, SYNAPSE effectively baits malicious actors who are likely to focus their efforts on probing and exploiting databases.

When a user or attacker connects to the SYNAPSE honeypot via SSH, the system begins by presenting a simulated Linux terminal environment, as described in earlier sections. Once inside, the MySQL server interface is triggered if the attacker attempts to access the service by entering a relevant command such as *mysql*. The user is then prompted to log in, as they would when connecting to a real MySQL server. The login prompt, authentication process, and subsequent interface closely mimic the behaviour of a legitimate MySQL server. However, behind the scenes, SYNAPSE leverages generative AI to simulate the entire environment. The user believes they are interacting with a fully functional MySQL service, but no database exists on the machine. Instead, the AI responds to user inputs by generating real-time database structures, table listings, query results, and other outputs. This dynamic generation ensures that attackers remain convinced they are interacting with an absolute MySQL service, encouraging them to continue their activities. Like the simulated terminal, nothing physically exists on the device; the entire interaction is a Python program controlled by SYNAPSE, with the AI generating all responses and content.

Interaction Flow

The interaction flow is the same as in Figure 3.3, with the simulated MySQL terminal replacing the Linux one. Upon entering the MySQL interface, users are prompted to log in as they would on a natural interaction. If the process is successful, SYNAPSE dynamically generates the connection ID and server version in the login message. These details are crafted to make the environment even more convincing, displaying a typical connection message like the one depicted in Figure 3.6.



```
enea@devstation:~$ mysql -ppassword
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12
Server version: 8.0.28-0ubuntu0.20.04.3 (Ubuntu)

Copyright (c) 2000, 2022, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their res
pective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> |
```

Figure 3.6. MySQL Login Message Example.

As the user interacts with the simulated MySQL service, SYNAPSE's generative AI interprets each SQL command and generates appropriate, coherent responses that mimic the behaviour of an actual MySQL database. These outputs, whether displaying databases, tables, or executing queries, are dynamically generated based on the attacker's input, maintaining the illusion of a live MySQL environment. Just like with the terminal simulation, each command and response is tracked by SYNAPSE, ensuring contextual consistency. For instance, if the attacker creates a database, it will remain in the system's memory and be accessible in subsequent queries. This ensures that repeated commands, such as *SHOW DATABASES*, return consistent results, further convincing the attacker of the system's legitimacy.

Prompt

The prompt simulating the MySQL service in SYNAPSE follows the same approach as the terminal simulation, explicitly instructing the AI to behave as a MySQL server running inside a Linux OS terminal. The key to this approach is carefully crafted prompt engineering, which directs the AI to generate responses that closely mimic those of an actual MySQL server. The initial part of the prompt includes the following instructions:

You are a MySQL server inside a Linux OS terminal. You act and respond exactly as a MySQL server. You will respond to all commands just as a MySQL server would ...

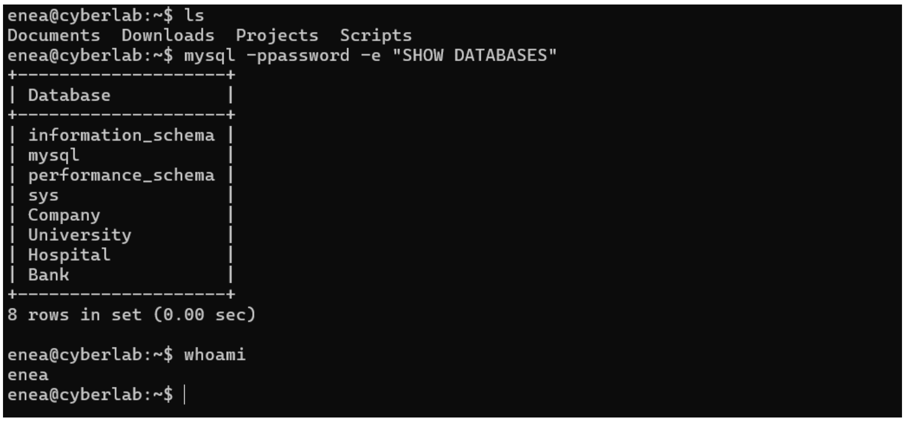
In addition to these basic instructions, many examples are provided to the AI to demonstrate how it should behave in various real-world scenarios. These examples teach the AI to handle common SQL commands such as *SHOW DATABASES*, *CREATE TABLE*, *INSERT INTO*, and *SELECT*. They are essential for guiding the AI in generating appropriate outputs that align with typical database interactions. Moreover, the AI is instructed to dynamically generate databases, tables, and columns, simulating content that would be expected in a professional IT environment. To make the honeypot even more attractive to attackers, the AI is also directed to include sensitive and enticing data, such as passwords or credit card numbers, in these dynamically generated tables. This approach heightens the appeal of the simulated MySQL service, encouraging attackers to delve deeper, believing they have uncovered valuable information.

Features

The MySQL service simulation in SYNAPSE is designed to provide a convincing and interactive experience while maintaining security and control. Several key features have been implemented to enhance the realism and effectiveness of the simulation.

To protect the honeypot and avoid exposing sensitive and fake data, the AI is instructed to block commands that could lead to the deletion of tables or unauthorised access to sensitive information. For instance, if an attacker attempts to use the *DROP TABLE* command or access sensitive columns like passwords or credit card numbers, the system responds by denying access or rejecting the command. This helps safeguard the environment and makes the simulation more convincing, as attackers may interpret these blocks as realistic security mechanisms, further engaging with the database and making additional attempts to exploit it. Like the terminal simulation, the MySQL service supports all common exit commands, including *exit*, *quit*, and *\q*. When a user or attacker issues any of these commands, the MySQL session is cleanly terminated, and the user is returned to the simulated Linux terminal environment. This smooth transition reinforces the authenticity of the simulation, maintaining continuity and ensuring that the interaction feels consistent with real-world behaviour.

The login process for the MySQL service has been meticulously crafted in Python. The system checks login arguments such as “-u” for the username and “-p” for the password, responding with appropriate error messages if something goes wrong, such as an invalid username or incorrect password. Moreover, SYNAPSE supports MySQL inline command execution using the “-e” option, just as a real MySQL server would. This allows attackers to execute MySQL commands directly from the terminal without logging into the MySQL interface. For instance, an attacker could run *mysql -u enea -ppassword -e ”SHOW DATABASES”* (see Figure 3.7) to view the list of databases in one step. This functionality, implemented through Python, enhances the flexibility of the simulation, giving attackers more options for interacting with the database and making the environment even more appealing for exploration.



```
enea@cyberlab:~$ ls
Documents  Downloads  Projects  Scripts
enea@cyberlab:~$ mysql -ppassword -e "SHOW DATABASES"
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| Company            |
| University         |
| Hospital           |
| Bank               |
+--------------------+
8 rows in set (0.00 sec)

enea@cyberlab:~$ whoami
enea
enea@cyberlab:~$ |
```

Figure 3.7. MySQL “-e” Option Example.

In the same way SYNAPSE dynamically generates a simulated filesystem for the terminal, the MySQL service simulation uses AI to dynamically create databases, tables, columns, and rows. There is no pre-existing database or static data structure; the AI creates realistic and plausible data based on user interactions. When a user or attacker runs commands to list databases, display table structures, or query data, SYNAPSE's AI generates the necessary elements in real-time. This includes dynamically creating database names, table schemas, and column definitions that align with the typical data an attacker might expect to find in a natural IT environment, as specified in the AI prompt. For example, upon issuing a *SHOW DATABASES* command, the AI produces a list of plausible database names, such as "employees", "clients", or "project_management". Similarly, when the user queries tables, the AI generates columns like "username", "email", "password", or "credit_card_number" to reflect sensitive information that typically attracts attackers. The row content is also dynamically generated to match the columns, ensuring the results appear consistent and believable.

Additionally, the AI's "memory" considerations for the MySQL service simulation are the same as those discussed earlier (see Section 3.1.6). The entire session history, including commands and responses, is retained to ensure consistency and continuity throughout the user's interaction with the system. Combined with the dynamic generation of content and responses, these features ensure that the MySQL service simulation in SYNAPSE appears realistic and offers attackers a robust and engaging experience. By controlling access to sensitive data and providing a convincing login and command execution environment, SYNAPSE effectively captures intruders' behaviour while keeping the system secure.

Additional Services

Integrating the MySQL service simulation into SYNAPSE provides a framework that can be extended to simulate other services in the future. Adding MySQL required crafting the necessary prompt to guide the AI's behaviour, implementing specific features and functions similar to the terminal simulation, and creating what can be termed "glue code" between the simulated Linux terminal and the MySQL service. This integration ensures a seamless and believable transition when a user or attacker switches from interacting with the terminal to accessing MySQL. The glue code manages how the MySQL service is invoked from within the terminal, handling command interpretation, login management, and the transition back to the terminal after executing MySQL commands. For instance, when a user types *mysql* into the terminal, the written code facilitates the shift from the terminal simulation to the MySQL simulation by ensuring that the appropriate prompt is sent to the AI and that the system responds as a MySQL server.

This approach to integrating MySQL can be generalised to add other services. Each new one would require its glue code to manage the interaction between the terminal and the service. Furthermore, specific code would need to be written to adapt the simulation, making it as realistic as possible by replicating the expected behaviour, responses, and functions. In summary, while the prompt engineering and features are critical for simulating a service, the glue code is equally essential for enabling smooth integration within SYNAPSE's overall system. As additional services are considered for future implementation, this same methodology, combining prompt engineering, feature creation, and glue code, will be essential to creating convincing simulations that further enhance the honeypot's ability to engage attackers.

3.2 Collected Data

Data extraction and analysis in cybersecurity have become essential for staying ahead of sophisticated threats. Honeypots are critical tools to collect real-time data on attacker behaviours, including the techniques, tactics, and procedures (TTPs) employed during intrusion attempts. The ability to harness this data effectively is critical to improving defence mechanisms. The honeypot data can be analysed dynamically and automatically with generative AI, which can process vast amounts of this information, identify patterns, generate insights, and adapt to new threat vectors. This leads to a more proactive cybersecurity posture, where real-time attack patterns can be identified, allowing quicker responses and more refined defensive strategies.

This section will describe the data extracted using SYNAPSE (see Figure 3.8 for an overview). Each collected dataset is tied to specific IP addresses, allowing for granular tracking of attacker activities. For every new connection from an IP address, that is the first session, comprehensive data is collected. Subsequent connections from the same IP address are tracked differently. Instead of duplicating the entire dataset, SYNAPSE increments the number of connections and logs the duration of each session beyond updating the logs. This ensures that critical details are captured during the initial interaction while keeping track of long-term engagement by monitoring reconnections. This approach allows for more efficient analysis, focusing on the initial attack details and the behavioural patterns of persistent or recurring attackers.

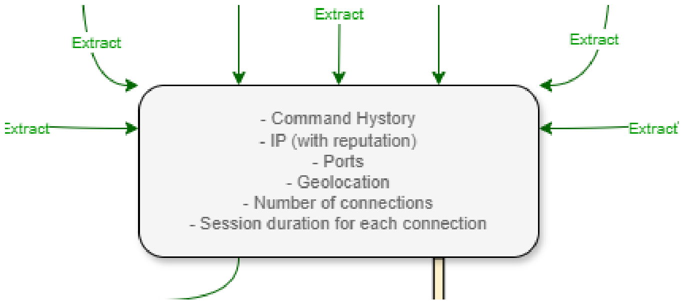
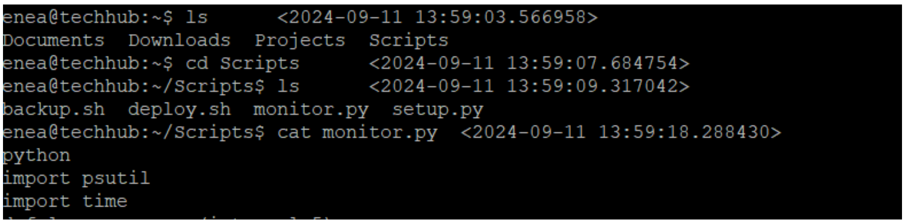


Figure 3.8. Collected Data Overview.

3.2.1 History of Commands and Responses

One of the first data types extracted for analysis in SYNAPSE is the **history of commands and responses**. Two separate histories are maintained for each IP address: a terminal history and a MySQL history. The terminal history records all interactions when the user engages with the terminal, while the MySQL history tracks interactions specific to the MySQL environment. Both logs are updated in real-time, recording the user's command and the corresponding date and appending the AI-generated response. This process facilitates tracking and supports the AI's "memory" functionality, as outlined in Section 3.1.6. These logs serve as a detailed command-response timeline for each IP address, allowing for an in-depth analysis of attacker behaviour, tactics, and strategies. The AI's memory mechanism leverages these histories to contextualise future responses based on past interactions, making SYNAPSE a more adaptive and reactive system in dealing with repeated engagements from the same user (see Figure 3.9 for an example).

A separate file is maintained for each session initiated by an IP address. This session-specific file contains logs exclusively for that session, keeping track of commands and responses within that isolated interaction. When the SYNAPSE-to-MITRE extension is activated, these session-specific logs are used to enhance analysis by mapping commands to known MITRE ATT&CK techniques, which will be explored in more detail in Section 3.3.



```
enea@techhub:~$ ls      <2024-09-11 13:59:03.566958>
Documents  Downloads  Projects  Scripts
enea@techhub:~$ cd Scripts      <2024-09-11 13:59:07.684754>
enea@techhub:~/Scripts$ ls      <2024-09-11 13:59:09.317042>
backup.sh  deploy.sh  monitor.py  setup.py
enea@techhub:~/Scripts$ cat monitor.py  <2024-09-11 13:59:18.288430>
python
import psutil
import time
```

Figure 3.9. Terminal History Example.

3.2.2 IP Address and IP Reputation

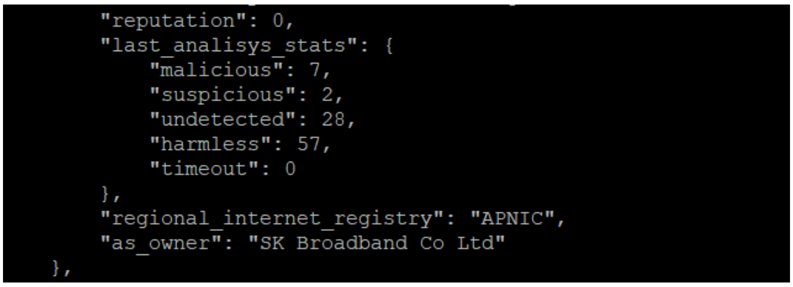
A key aspect of SYNAPSE's data extraction process involves tracking and managing **IP addresses** for each user. When a user connects to the system for the first time, the IP address is extracted from the *SSH_CLIENT* environment variable. This IP is then appended to a newly created data file explicitly dedicated to that user, ensuring that all information is appropriately organised and logged under the correct identifier. A directory named after the IP address is automatically created upon the first connection. It is a container for all the data files associated with that particular IP, ensuring that each user's interactions and activities are isolated and neatly organised. All subsequent files related to the user's sessions, such as command histories, MySQL interactions, and session-specific logs, will be stored within this directory. The files themselves are named by prepending the IP address to the filename, further simplifying the structure and allowing for easy retrieval and analysis. Beyond organisational purposes, the IP address is displayed in the login message whenever the user connects via SSH (see Section 3.1.4).

Additionally, the extracted IP address can be cross-referenced with external threat intelligence databases to assess if it is linked to known malicious activity. This enables the dynamic enrichment of SYNAPSE's data by integrating external insights, further improving the system's defensive and analytical capabilities. In particular, the corresponding **IP reputation** is extracted along with the IP address. This is accomplished by leveraging the VirusTotal APIs[4], a widely-used platform in cybersecurity that aggregates data from various antivirus engines, URL scanning services, and other threat analysis tools to provide comprehensive threat intelligence. VirusTotal enables users to upload files, URLs, and IP addresses to assess whether they are associated with known malware, phishing attempts, or other malicious activities. When SYNAPSE cross-references an IP address with VirusTotal, it extracts the reputation score and detailed information about the IP's online behaviour and potential risks. The critical data extracted from VirusTotal includes (see Figure 3.10 for an example):

- **total_votes**: represents the total votes from different sources in VirusTotal regarding whether the IP address is considered malicious or harmless. It provides a broad view of how the IP is perceived within the threat intelligence community.
- **whois**: provides ownership and registration information about the IP address, which can help identify the entity or organisation behind the IP.
- **reputation**: a numerical score that reflects how trustworthy or suspicious the IP address is, based on VirusTotal's analysis and threat detection.
- **last_analysis_stats**: gives an overview of the most recent IP analysis results, including the number of engines that flagged it as malicious.
- **regional_internet_registry**: information about the regional registry responsible for managing the IP address allocation. This can offer geographic insights into the IP's origin.
- **as_owner**: the Autonomous System (AS) owner provides details about the network that the IP is part of, which can further identify potential threat sources or explain network behaviour.

The reputation of an IP address is precious in cybersecurity for several reasons. First, if an IP has been flagged in previous cyber incidents, knowing its reputation can immediately signal the potential risk of the user behind that IP. This allows SYNAPSE to take preemptive measures or flag the interaction for further scrutiny. An IP address with a poor reputation (e.g., associated with botnets, malware, or phishing) provides important context about the likely intentions of the user. This information can help prioritise investigations and response efforts. Then, security analysts can make more informed decisions by combining the IP's reputation with SYNAPSE's collected data, such as commands, responses, and session details. This integration helps create

[4] https://www.virustotal.com/gui/home/upload



```
        "reputation": 0,
        "last_analisys_stats": {
            "malicious": 7,
            "suspicious": 2,
            "undetected": 28,
            "harmless": 57,
            "timeout": 0
        },
        "regional_internet_registry": "APNIC",
        "as_owner": "SK Broadband Co Ltd"
    },
```

Figure 3.10. IP Reputation Example.

a fuller profile of the potential attacker and their behaviours. Lastly, when integrated into a system like SYNAPSE, the reputation score can trigger automated responses, such as isolating or blocking users from high-risk IPs, thus enhancing the overall security posture.

3.2.3 Ports

In addition to the IP address, **ports** are extracted from the *SSH_CLIENT* environment variable during each connection. The source and destination ports are crucial in understanding the context of communication between the user and the system. When a user connects, SYNAPSE logs the port information, including the source port, the one used by the user's machine to initiate the connection, and the destination port, the one on the SYNAPSE server that the user is trying to access. This data is stored alongside the IP address in the user-specific data file and is used to enrich the analysis of each session.

By capturing port data, SYNAPSE can identify unusual or suspicious activity, such as connections to uncommon ports or repeated targeting of specific services. This helps detect potential attack vectors and informs analysts about which services may be at risk. Additionally, SYNAPSE can use this data to correlate connection attempts with known attack patterns, enhancing its ability to monitor, analyse, and respond to threats in real-time.

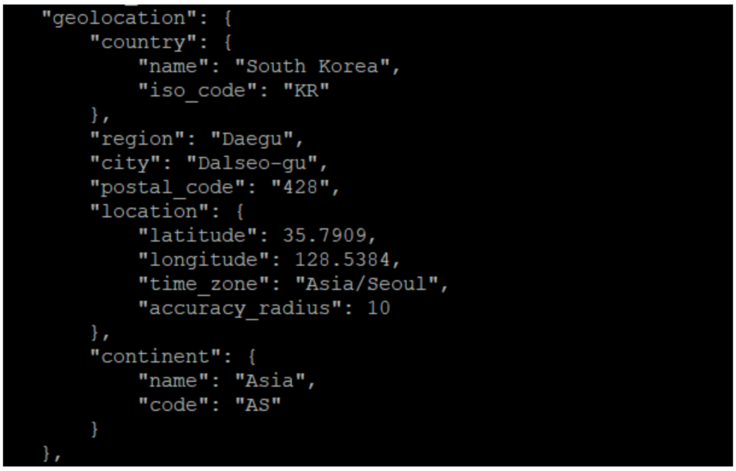
3.2.4 Geolocation

SYNAPSE incorporates **geolocation** data as part of its threat analysis by leveraging the GeoLite2 database[5] to map IP addresses to geographic locations. A cron task is scheduled to update this database daily at midnight, ensuring the system has the latest geolocation information. This helps maintain accurate location tracking for all connections, improving the quality of data analysis and security insights. When a user connects, several critical geolocation details are extracted based on the IP address (see Figure 3.11 for an example):

- **Country**: the name and the ISO code of the country where the IP address is registered. This helps quickly identify the connection's national origin and can be used to assess the likelihood of malicious intent, especially if the connection comes from regions known for cybercriminal activity.
- **Region and City**: the specific region and city associated with the IP address. This level of detail allows analysts to narrow down the exact origin of the connection, providing a clearer understanding of the geographic patterns in user behaviour.
- **Postal Code**: the postal or ZIP code for the location, which can add a layer of precision to the geographical data. This can be particularly useful in highly localised analysis, for example, investigating if repeated attacks come from a narrow geographic area.

[5] https://dev.maxmind.com/geoip/geolite2-free-geolocation-data

- **Location Coordinates**: the latitude and longitude of the IP address provide precise mapping coordinates, making it possible to plot the connection's origin on a map. This is useful in visualising attack patterns or connections across different regions.
- **Time Zone**: the time zone of the IP address, which helps align connection times with local time frames. This can be valuable when tracking the timing of attacks and seeing if they correspond to typical working hours in the attacker's location.
- **Accuracy Radius** indicates how accurate the geolocation data is, approximating how close the detected location is to the user's real-world one. A large accuracy radius may indicate that the exact location is less specific, while a smaller radius suggests higher precision.
- **Continent**: the continent name and its code are extracted to provide a broader geographic context. This can be used for high-level trend analysis, such as monitoring attacks from specific continents or regions.



```
    "geolocation": {
        "country": {
            "name": "South Korea",
            "iso_code": "KR"
        },
        "region": "Daegu",
        "city": "Dalseo-gu",
        "postal_code": "428",
        "location": {
            "latitude": 35.7909,
            "longitude": 128.5384,
            "time_zone": "Asia/Seoul",
            "accuracy_radius": 10
        },
        "continent": {
            "name": "Asia",
            "code": "AS"
        }
    },
```

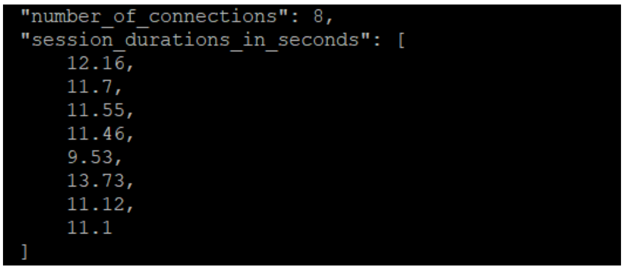
Figure 3.11. Geolocation Information Example.

Geolocation data offers significant value in understanding the origins and behaviours of users or attackers interacting with SYNAPSE. This information adds geographic context to the connection, allowing security analysts to make more informed decisions when assessing the risk level of each session. For instance, connections from specific countries or regions known for frequent cyberattacks may raise immediate flags, allowing the system to apply more scrutiny or restrictions to those sessions. On the other hand, sudden changes in the geolocation of a returning user, such as an IP that previously originated from one region but now shows a different, distant location, could indicate the use of anonymisation techniques like VPNs or proxies, suggesting an attempt to obscure the user's actual location. Ultimately, including geolocation data enhances SYNAPSE's overall analytical capabilities, allowing for more nuanced threat detection and a better understanding of attack patterns on both a local and global scale.

3.2.5 Number of Connections and Duration for Each Connection

SYNAPSE keeps track of the **number of connections** from each IP address and the **duration of every session**. Each time a user or attacker connects from the same IP address, SYNAPSE increments the connection count and records the duration of the session in seconds (see Figure 3.12 for an example).

Tracking the number of connections allows SYNAPSE to monitor the persistence of an attacker. Frequent reconnections from the same IP address may indicate a determined malicious actor actively probing the system for vulnerabilities or attempting to establish a foothold. This behaviour suggests that the attacker views the honeypot as a potentially valuable or vulnerable



```
"number_of_connections": 8,
"session_durations_in_seconds": [
    12.16,
    11.7,
    11.55,
    11.46,
    9.53,
    13.73,
    11.12,
    11.1
]
```

Figure 3.12. Number of Connections with Durations Example.

target, providing more opportunities to capture data about their tactics and methods. The duration of each session offers further insights into user behaviour. Longer sessions can imply that the attacker is engaged and believes the honeypot is a legitimate system, making them more likely to interact extensively with it. This can result in more data being collected, including detailed command histories and interaction patterns, which are invaluable for analysis. On the other hand, short sessions may indicate that the attacker quickly realised the system was a honeypot or failed to find any exploitable weaknesses.

In addition to monitoring individual attacker behaviour, the combination of connection count and session duration provides a way to assess the attractiveness of the honeypot itself. If attackers frequently return and stay connected for long periods, it suggests that the honeypot is well-designed and effective in luring malicious actors. This data can also help identify trends or patterns, such as recurring connection attempts at specific times, which might point to organised efforts or automated attacks. Similarly, many rapid, short-duration connections could signal the use of automated tools or bots, offering insights into different types of threats and allowing for adjustments in defensive strategies.

3.3 SYNAPSE-to-MITRE

In the current cybersecurity landscape, mapping adversarial tactics and techniques to a standardised framework has become increasingly vital for defence and threat detection. One of the most widely adopted frameworks is the MITRE ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) knowledge base, which categorises and details cyber adversaries' behaviour, tools, and strategies.

SYNAPSE-to-MITRE is a critical extension within the SYNAPSE project, enabling automated mapping of the logs and data collected by the honeypot to the MITRE ATT&CK database (see Figure 3.13 for an overview). This integration enhances the identification of known attack patterns, thus providing a structured way of understanding the adversary's actions based on real-time data generated by the honeypot. Mapping the data to MITRE ATT&CK facilitates a more comprehensive attack analysis and enables better decision-making for incident response and threat mitigation strategies. The importance of this mapping lies in the fact that the framework provides a common language and structure for documenting attack behaviours, making it easier for security teams to collaborate and share information. Then, by correlating SYNAPSE's captured data with the MITRE framework, security analysts can more quickly identify the techniques used and understand the likely goals of the attacker. Lastly, the integration helps predict future attack vectors by analysing trends in the captured data and comparing them against known adversarial tactics.

3.3.1 Genesis: from cti-to-mitre to SYNAPSE-to-MITRE dataset

The SYNAPSE-to-MITRE extension leverages machine learning to map SYNAPSE logs to the MITRE ATT&CK framework. Central to this approach is a machine learning model that must be trained on a carefully curated dataset to ensure accurate and reliable threat detection and

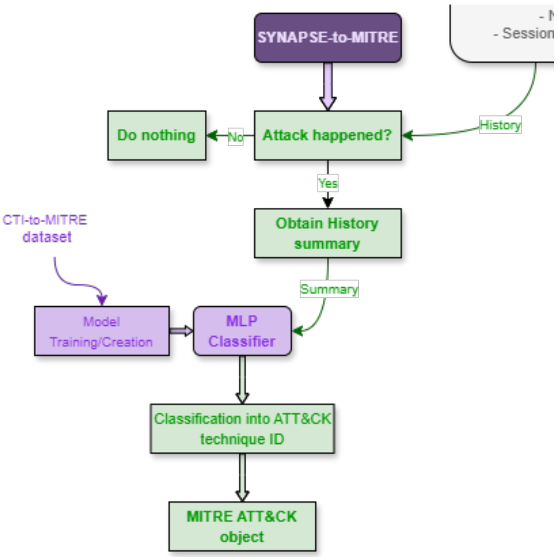
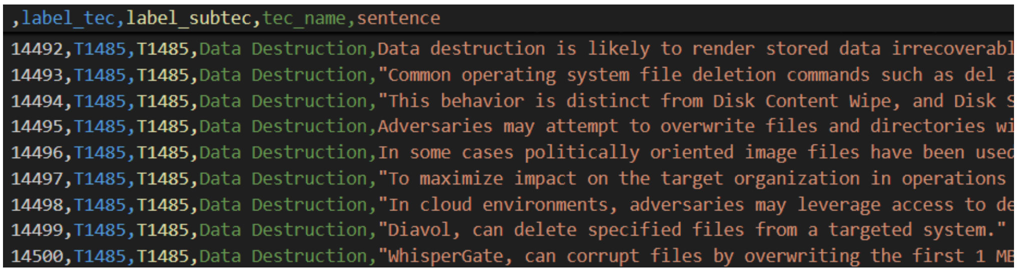


Figure 3.13. SYNAPSE-to-MITRE Overview.

classification. In this research, the dataset used is sourced from the work developed by DESSERT Lab, titled “CTI-to-MITRE with NLP” [21]. This dataset comprises cybersecurity threat intelligence (CTI) reports pre-processed and mapped to the MITRE ATT&CK framework using natural language processing (NLP) techniques. These reports include detailed descriptions of observed attacks, the tactics and techniques employed, and contextual metadata that assists in mapping each threat to the corresponding classification. The dataset is built by leveraging the MITRE ATT&CK and Common Attack Pattern Enumeration and Classification (CAPEC) knowledge bases, distributed in the Structured Threat Information Expression (STIX) format. They provide a comprehensive and standardised taxonomy of known attack techniques and patterns, which serve as the foundation for the dataset. The latter is organised in a structured format that includes the following key fields (see Figure 3.14 for an example):

- **Technique Label**: identifies the primary MITRE ATT&CK technique using a unique identifier (e.g., T1055) corresponding to specific adversarial tactics or techniques.
- **Sub-Technique Label**: specifies the sub-technique associated with the primary technique (e.g., T1055.011), capturing variations of the attack method.
- **Technique Name**: a human-readable name for the technique or sub-technique (e.g., “Extra Window Memory Injection”), making the taxonomy more accessible.
- **Sentence**: each row in the dataset contains a detailed sentence that describes various aspects of the attack technique or sub-technique, such as how it operates, its purpose, or the vulnerabilities it exploits.



```
,label_tec,label_subtec,tec_name,sentence
14492,T1485,T1485,Data Destruction,Data destruction is likely to render stored data irrecoverabl
14493,T1485,T1485,Data Destruction,"Common operating system file deletion commands such as del a
14494,T1485,T1485,Data Destruction,"This behavior is distinct from Disk Content Wipe, and Disk S
14495,T1485,T1485,Data Destruction,Adversaries may attempt to overwrite files and directories wi
14496,T1485,T1485,Data Destruction,In some cases politically oriented image files have been used
14497,T1485,T1485,Data Destruction,"To maximize impact on the target organization in operations 
14498,T1485,T1485,Data Destruction,"In cloud environments, adversaries may leverage access to de
14499,T1485,T1485,Data Destruction,"Diavol, can delete specified files from a targeted system."
14500,T1485,T1485,Data Destruction,"WhisperGate, can corrupt files by overwriting the first 1 MB
```

Figure 3.14. Dataset Example.

This dataset allows the system to start from unstructured CTI sentences, which typically contain descriptions of observed threats and attacks, and map them to the corresponding MITRE ATT&CK techniques and sub-techniques. By doing so, the machine learning model learns to associate specific patterns in log data with known attack tactics, enabling the automatic classification of new, incoming logs into the appropriate MITRE ATT&CK categories. By building on the MITRE ATT&CK and CAPEC knowledge bases, this dataset ensures that the mapping of CTI sentences is grounded in established, industry-recognised standards. This structured transformation from unstructured threat intelligence data into actionable MITRE mappings significantly enhances the SYNAPSE-to-MITRE system's ability to detect and categorise cyber threats while aligning with widely accepted cybersecurity frameworks.

Updating the Dataset

Despite the "CTI-to-MITRE with NLP" robust and well-structured dataset, it was somewhat outdated because it aligned with MITRE ATT&CK version 10.1. To address this issue, the SYNAPSE-to-MITRE extension re-built the dataset using the original code provided by DESSERT Lab. It has been updated to incorporate all the knowledge from the most recent enterprise-attack version, **15.1**. The benefits of updating the dataset to the latest release are significant. The model gains a more comprehensive understanding of current attack methods by including all the new techniques and sub-techniques introduced in recent MITRE ATT&CK updates. This enables more accurate mapping of CTI sentences to the correct MITRE techniques, improving the precision of threat detection and classification. The updated dataset encompasses a broader range of attack tactics and strategies, allowing SYNAPSE-to-MITRE to recognise and respond to more recent and sophisticated cyber threats. This expanded coverage ensures that the system can detect emerging attack vectors that may not have been captured in older versions of the ATT&CK framework. Moreover, using the most recent version of the MITRE ATT&CK framework ensures the system remains aligned with the latest industry standards. This is crucial for maintaining relevance in a rapidly evolving cybersecurity environment, as organisations and security professionals rely on the MITRE framework to stay updated on the newest threat techniques. The updated dataset allows the machine learning model to train on more diverse and relevant data, which improves its ability to generalise to new, unseen logs.

However, with the inclusion of the expanded dataset, there is a trade-off in model accuracy. As the dataset is significantly more prominent due to the additional techniques and sub-techniques introduced in version 15.1, the complexity of the classification task has increased. As a result, the accuracy of the machine learning model shows a slight decrease compared to the evaluations conducted by DESSERT Lab, which used the smaller dataset aligned with enterprise-attack version 10.1. The larger one presents a broader array of techniques for the model to learn and classify, which, while increasing the coverage of potential threats, introduces more variability and thus slightly impacts overall accuracy. Despite this decrease, the benefits of the larger, more up-to-date dataset outweigh the trade-off, as the system is now capable of recognising and responding to a far greater number of attack techniques, many of which reflect the most recent developments in cyber threats.

3.3.2 Model Training

The SYNAPSE-to-MITRE extension trains a machine learning model using the updated dataset derived from the MITRE ATT&CK framework, leveraging the code provided by the DESSERT Lab. The machine learning model employed in this training process is a Multilayer Perceptron (MLP) classifier, implemented using "scikit-learn"[6], a widely-used machine learning library in Python. Scikit-learn offers efficient tools for data mining, data analysis, and machine learning, with simple and reusable code, making it an ideal choice for rapid model development. The MLP classifier is particularly effective for classification tasks that map input data, like unstructured CTI

[6] https://scikit-learn.org/stable/

sentences, to a predefined set of classes, like MITRE ATT&CK techniques and sub-techniques. Other machine learning models, such as Random Forests and Support Vector Machines (SVM), were also explored during experimentation. However, none of these models achieved the same accuracy and performance as the MLP classifier, reinforcing its suitability for this classification task. After training the model on the expanded dataset, the F1 score, a performance metric that balances precision and recall, was calculated to be 65%. This relatively low result can be attributed to the size and complexity of the updated dataset, which includes significantly more techniques and sub-techniques from MITRE ATT&CK version 15.1 compared to the earlier version.

The SYNAPSE-to-MITRE model was adapted to improve performance. Instead of predicting a single MITRE ATT&CK mapping for each unstructured CTI sentence, the model outputs the top three possible mappings. This approach boosts the model's F1 score to **89%**, significantly improving its classification accuracy by increasing the likelihood that the correct mapping is included within the top three choices. This strategy offers both advantages and drawbacks. The chances of including the correct mapping increase, leading to a higher F1 score. This makes the model more reliable in scenarios where precision is critical, such as in security operations where analysts need accurate classifications to detect and respond to threats. The broader output scope reduces the likelihood of missing the correct technique, ensuring the system doesn't overlook critical threats. This is also particularly useful in complex cases where a CTI sentence may correspond to more than one valid attack technique. However, while the system offers three possible mappings, security analysts may need to verify and choose the most accurate mapping manually. This additional step introduces more cognitive load and can slow decision-making, especially in high-pressure environments. Sometimes, offering multiple potential mappings might create ambiguity, mainly when the difference between the predicted techniques is subtle. This could lead to indecision or incorrect conclusions if the analyst cannot accurately assess which of the three predictions is most relevant. While the three-mapping approach increases the model's overall performance, it comes with the trade-off of requiring human interpretation to finalise the correct MITRE mapping in many cases. This balance between automation and human oversight must be carefully managed in practical applications.

Test Set vs Real-World Data

It is important to note that the F1 scores reported, both 65% and the improved 89%, are calculated based on test datasets, not real-world data. Test datasets are typically curated and cleaned, ensuring that the data quality is high and that the distribution of classes is representative of the dataset as a whole. These conditions are often not reflective of real-world scenarios. In practical settings, where accurate, unstructured CTI data is fed into the model, the F1 score is expected to decrease. This is due to several factors. First, real-world CTI data is often noisy, incomplete, or imprecise, which makes it more difficult for the model to extract relevant features and make accurate predictions. Then, the dataset used to train the model is based on the known threat landscape as captured in the MITRE ATT&CK framework. However, in the dynamic world of cybersecurity, new tactics, techniques, and procedures (TTPs) constantly emerge, meaning the model may struggle to classify unseen or novel threats. Moreover, some attack techniques may be far more common than others, leading to class imbalance. While this may not significantly impact the test dataset, it can reduce the model's performance on actual data, where specific rare techniques may be underrepresented or overrepresented. When applied to real-world data, the overall decline in F1 score is a common challenge in machine learning, especially for security-related tasks. However, the approach taken by SYNAPSE-to-MITRE helps mitigate this decline by increasing the flexibility of predictions, even if real-world performance is not as high as the test results suggest. Future work may involve refining the dataset or model to improve performance on real data further, ensuring that the system remains effective in practical applications.

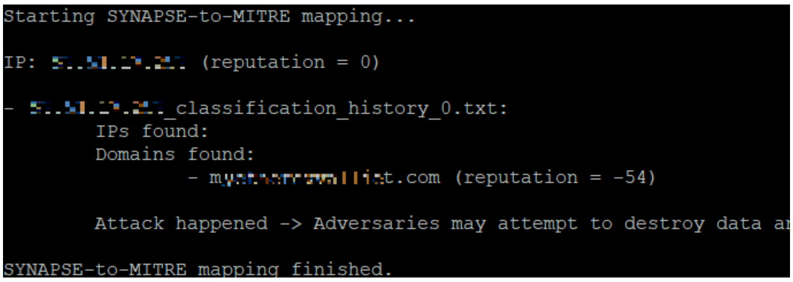
3.3.3 Mapping Workflow

The mapping workflow consists of several vital phases that guide the process from initial log file analysis to final attack mapping. Each stage contributes to the system's ability to provide structured, actionable intelligence, ensuring that detected threats are accurately mapped and stored for future analysis. Now, each phase is described in detail.

Phase one: Attack Decision and Reputation Analysis

The workflow begins by activating the core script of the SYNAPSE-to-MITRE extension. Once started, it initiates a process where each folder corresponding to a unique user is scanned. These folders are organised by IP address and contain session logs documenting interactions between a specific user and SYNAPSE. Each log file represents a distinct connection session, capturing the commands and responses exchanged during that interaction (see Section 3.2.1).

The AI processes each log file, examining the user's actions in that particular session. Its primary goal is to determine whether any malicious activity has occurred. The AI is prompted with all the log file contents and is tasked with assessing whether the behaviour constitutes an attack, adopting "Few-Shot Prompting" in the prompt used to feed it. To improve the accuracy of this assessment, the AI also analyses the reputation of the IP addresses and domains involved in the session. First, the reputation of the user's IP address is checked using methods outlined in Section 3.2.2. This step allows the system to gain insights into whether the user has a known history of malicious activity. Next, the entire log file is scanned to search for any IP addresses or domains the user may have interacted with during the session. For instance, if a user attempts to connect to an external server via SSH, the system checks the reputation of the server's IP address. Suppose this server has a poor reputation, as identified by security services like VirusTotal. The session likely involved malicious activity in that case, even if the individual actions seemed benign. The exact process is applied to domain names; if a user acts like downloading a file from a suspicious website using a command such as *wget*, the AI will flag the interaction as suspicious based on the domain's reputation. By leveraging reputation data from third-party services, the AI can better distinguish between regular and malicious interactions, even when individual actions might not immediately raise red flags. This multi-layered analysis strengthens the system's ability to identify potential threats precisely and informally (see Figure 3.15).



```
Starting SYNAPSE-to-MITRE mapping...

IP: [redacted] (reputation = 0)

- [redacted]_classification_history_0.txt:
        IPs found:
        Domains found:
                - my[redacted]t.com (reputation = -54)

        Attack happened -> Adversaries may attempt to destroy data a

SYNAPSE-to-MITRE mapping finished.
```

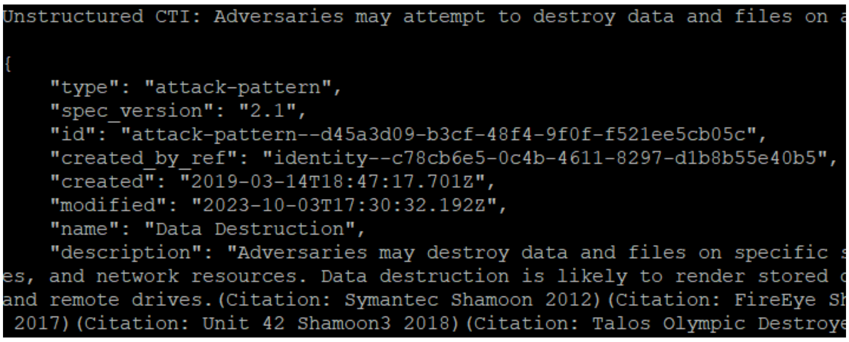
Figure 3.15. SYNAPSE-to-MITRE Workflow Example.

Phase two: Sentence Generation

Once the AI has evaluated the log file and determined no attack has occurred, the process ends, and no further actions are taken. However, if an attack is detected, the system transitions to the next workflow phase. The entire log file is once again prompted to the LLM. This time, the latter is tasked with generating a concise sentence that accurately describes the nature of the attack. This step aims to produce a summary that aligns closely with the structure of sentences found in the training dataset used by SYNAPSE-to-MITRE. The dataset consists of brief, well-structured descriptions of various attack techniques, which the model utilises to map unstructured log data to specific techniques and sub-techniques from the MITRE ATT&CK framework. A "Few-Shot Prompting" approach is again employed to ensure the AI-generated sentence adheres to the expected format. This method involves providing the AI with examples that illustrate the desired sentence structure, helping it model the output accordingly. The prompting guides the AI in producing sentences that match the brevity and style found in the dataset, making the generated descriptions as similar as possible to those used in it, allowing the model to maintain high accuracy when mapping the log file.

Phase three: Mappings Extraction

The mapping begins once the AI concisely describes the detected attacks. The previously trained model is loaded at this stage, and the generated sentence is passed. As described earlier, the model performs classification and outputs the top three potential mappings. These mappings are attack IDs corresponding to techniques or sub-techniques from the MITRE ATT&CK framework. However, the attack IDs are insufficient for practical use, as they are merely identifiers. The "mitreattack" Python library[7] is employed to provide more actionable insights and enhance the automation of the system's operations. It allows for the extraction of comprehensive information about each attack based on its unique ID. For each of the three attack IDs produced by the model, the corresponding ATT&CK object is retrieved using the library. Each contains detailed information about the attack technique, including its description, associated tactics, mitigations, and examples of use in real-world scenarios. This enriched data provides valuable context, making the output more useful for analysts who may need to investigate further or respond to the detected threat. Once the three ATT&CK objects are extracted, they are stored and organised. The results are saved to a directory with the original log file, ensuring that all relevant information is preserved for future analysis. By storing the results in this way, SYNAPSE-to-MITRE ensures that the detailed context of the detected attack is easily accessible for cybersecurity professionals who may need to review the findings or conduct deeper investigations into the event.



```
Unstructured CTI: Adversaries may attempt to destroy data and files on

{
    "type": "attack-pattern",
    "spec_version": "2.1",
    "id": "attack-pattern--d45a3d09-b3cf-48f4-9f0f-f521ee5cb05c",
    "created_by_ref": "identity--c78cb6e5-0c4b-4611-8297-d1b8b55e40b5",
    "created": "2019-03-14T18:47:17.701Z",
    "modified": "2023-10-03T17:30:32.192Z",
    "name": "Data Destruction",
    "description": "Adversaries may destroy data and files on specific
es, and network resources. Data destruction is likely to render stored
and remote drives.(Citation: Symantec Shamoon 2012)(Citation: FireEye Sh
 2017)(Citation: Unit 42 Shamoon3 2018)(Citation: Talos Olympic Destroye
```

Figure 3.16. SYNAPSE-to-MITRE ATT&CK Object Mapping Example.

3.4 Case Studies

Case studies are a powerful research tool used to explore real-world examples in depth, providing valuable insights into the practical application of theories and technologies. In cybersecurity, they are handy for evaluating system performance, identifying vulnerabilities, and comparing different approaches. In the context of SYNAPSE, a dynamic honeypot, case studies offer a valuable way to explore how advanced technologies interact in real-world scenarios. By focusing on two key implementations, DENDRITE, a static honeypot designed for comparison, and an AI vs. AI setup, where artificial intelligence is used to interact and attack SYNAPSE, we can gain deeper insights into the strengths and weaknesses of the project. These case studies provide a practical framework for understanding how SYNAPSE responds to traditional and AI-driven threats, highlighting its capabilities in adapting to evolving attack strategies. For additional insights into a less critical case study regarding analysing system logs leveraging AI, please refer to Appendix B.

[7] https://github.com/mitre-attack/mitreattack-python

3.4.1 DENDRITE

DENDRITE is a foundational case study comparing traditional static honeypots with more advanced, dynamic systems like SYNAPSE. Designed as a static honeypot, DENDRITE replicates the core functionalities of SYNAPSE as closely as possible, providing an identical classification and almost an equal feature set. This intentional similarity allows for a meaningful comparison between the two, isolating the critical differences in how static and dynamic honeypots respond to cybersecurity threats (see Figure 3.2 for an overview).

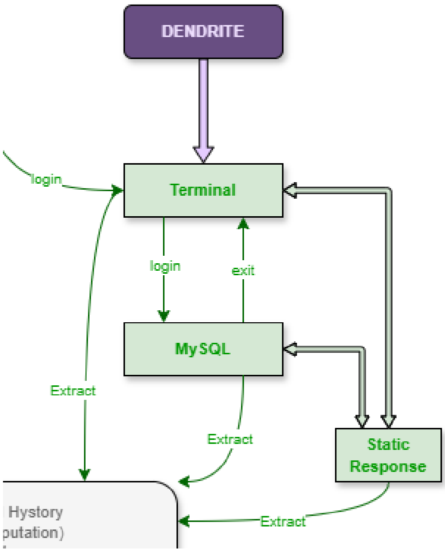


Figure 3.17. DENDRITE Overview.

Differences from SYNAPSE

While DENDRITE replicates many of the core functionalities of SYNAPSE, the two systems diverge significantly in their implementation and operational design, highlighting the contrasting advantages and limitations of the two different technologies. The primary distinction lies in their dynamic versus static nature. As a dynamic honeypot, SYNAPSE is integrated with generative AI, allowing it to adapt and modify its environment in response to external interactions. For example, SYNAPSE's filesystem is populated dynamically, enabling it to present an ever-changing environment that can effectively deceive attackers. DENDRITE, in contrast, operates with a prepopulated filesystem. While functional, this static approach lacks the adaptive capabilities of SYNAPSE, as the environment remains fixed and predictable once set.

Despite this difference, both honeypots extract the same user data. DENDRITE collects the same logs, user IP addresses, ports, geolocations, numbers of connections, and durations for each session from attackers as SYNAPSE does, ensuring that the key data points for analysis remain consistent across both systems. This enables a fair comparison of their performance despite the differences in their underlying architectures. Features such as tab autocompletion and sudo commands are similarly restricted in both honeypots, providing realistic interaction for the attacker. However, the mechanisms behind them differ. From a configuration perspective, DENDRITE offers a more straightforward setup. Since it leverages an actual OS and services, there is no need to simulate behaviours as SYNAPSE does. For example, DENDRITE uses an actual MySQL service with pre-inserted content that mirrors SYNAPSE's generated data. This simplifies DENDRITE's operation but at the cost of flexibility, as it cannot adapt its responses dynamically like SYNAPSE.

A key point of divergence, and one that leads into the next section, is the significant difference in resource requirements. DENDRITE is much heavier than SYNAPSE, relying on the Docker[8] ecosystem and containers to run its services and filesystem. While powerful, this containerised setup demands considerably more computational resources than SYNAPSE's lightweight design, which operates with just a running Python script and textual files. This difference in resource usage marks a crucial trade-off between the two honeypots, as SYNAPSE's efficiency and adaptability come at a fraction of the cost required to maintain DENDRITE.

Containers as User Environments

DENDRITE leverages Docker to isolate each user or attacker within a **containerised environment**, ensuring a secure setup. A specifically crafted Dockerfile was developed to recreate the necessary filesystem structure for the honeypot, tailored to mimic a natural Linux system for the targeted user interactions. The base image used is the latest version of Ubuntu, with the filesystem pre-populated with static, pre-crafted files to simulate a realistic user environment. Additionally, a MySQL service is installed and pre-populated with content that aligns with SYNAPSE's configurations, ensuring consistency between the two honeypots.

When a user or attacker connects to DENDRITE, their IP address is analysed to determine whether it is their first interaction. If true, a new Docker container is created specifically for that IP and runs the above image, establishing an isolated environment. Each container is assigned pre-set memory and CPU limits to prevent excessive resource consumption and ensure that the honeypot system remains operational even during high traffic or intensive attacks. If the IP has connected before, DENDRITE resumes the previously created and stopped container, allowing the user or attacker to pick up where they left off. This flow ensures efficient resource usage: when a user connects, the appropriate container is either run or started, the interaction is performed, and upon exiting, the container is stopped to conserve system resources.

This approach provides two fundamental benefits. First, it isolates the user or attacker entirely within a Docker container, preventing access to the host machine's real filesystem and enhancing the honeypot system's security. Second, it allows for session memory and resumption, similar to SYNAPSE's logging mechanism. In DENDRITE, each user has their own Docker container, comparable to how each user in SYNAPSE has their log file. This setup allows multiple sessions to be handled efficiently, enabling the attacker to resume their activities if they reconnect, as their previous session state is preserved within their container.

3.4.2 AI vs (SYNAPSE) AI

The AI vs AI use case explores how the "gpt-4o" generative AI interacts with the SYNAPSE dynamic honeypot. This scenario collects data and evaluates SYNAPSE's ability to handle advanced AI-driven interactions. By analysing this use case, insights can be drawn regarding SYNAPSE's capacity to adapt and respond to complex behaviours, particularly in scenarios involving AI-based threats.

Basic Interaction

In the basic interaction case, the AI establishes a connection with SYNAPSE through the SSH protocol, using "Paramiko"[9], a Python library that facilitates secure connections. Paramiko allows for the programmatic management of SSH sessions, enabling the AI to interact with SYNAPSE like a real-world user or attacker accessing a Linux environment, thus providing the AI with a realistic interaction framework.

Upon connecting, the artificial intelligence generates commands to interact with what it perceives as an actual Linux OS terminal, including access to the filesystem and a MySQL service.

[8] https://www.docker.com/

[9] https://github.com/paramiko/paramiko

These commands are produced based on the AI's understanding of a typical Linux environment, believing it executes standard operations within the OS. On the other side, SYNAPSE, designed to simulate a legitimate environment, processes these commands and responds as it would to any user, maintaining the illusion of interacting with a natural system. This setup creates an interesting dynamic: the AI generates commands as though they are interfacing with an actual OS, while SYNAPSE, unaware that the user is an AI, continues to engage with the interaction as if it were a regular user session. This introductory scenario is crucial for evaluating how SYNAPSE handles non-malicious, complex behaviours initiated by an AI, providing insights into its adaptability and response mechanisms in a controlled environment.

Attacker Interaction

The attacker interaction follows the same principles as the basic one but shifts focus from benign engagement to an attempt by the AI to breach and disrupt the SYNAPSE environment. Leveraging Paramiko to establish an SSH connection, the AI assumes the role of a malicious user or attacker, interacting with SYNAPSE as though it were an actual Linux OS terminal. In this scenario, however, the AI's goal is to explore or interact with the system and actively attempt to break or corrupt both the filesystem and the MySQL service. The AI generates a series of commands to damage or exploit the system, unaware that it operates within a controlled, simulated environment. The number of attacks the AI performs is configurable, defined by a predetermined value N. Once the experiment starts, the AI executes N attacks, stopping each one when it determines that it has completed its action. After reaching the limit N, the script automatically stops, concluding the study. During these interactions, SYNAPSE, which continues to perceive the AI as a real user, responds to the malicious commands as it would in an actual attack scenario.

This attacker interaction is essential for evaluating SYNAPSE's performance, as it allows for the analysis of how well the system defends itself against AI-driven intrusions. By conducting numerous automated attacks, this study provides valuable data on the types of interactions the AI can generate and SYNAPSE's ability to resist, mitigate, or adapt to them. This data is crucial for understanding the system's overall resilience and effectiveness in real-world scenarios, helping to inform future improvements and optimisations of SYNAPSE.

Chapter 4

Results

This chapter presents the results obtained by SYNAPSE during various experiments aimed at analysing human and AI-driven interactions. They were designed to evaluate SYNAPSE's effectiveness in detecting and mapping cyberattacks within the MITRE ATT&CK framework and understand its comparative performance against traditional honeypot systems. The first set of results will examine data from real-world attackers connected to SYNAPSE, analysing extracted information such as IP addresses and reputation, country, region, city, continent, time zone, ports used, and the number and duration of connections. This data offers a global perspective on the types of attacks and their geographical distribution. The second section will focus on controlled user experiments, where individuals with different levels of expertise interacted with both SYNAPSE and DENDRITE. By analysing their feedback and interactions, we will assess SYNAPSE's performance, including its ability to differentiate between malicious activities and legitimate behaviour. This will help determine how SYNAPSE performs when used in parallel with a traditional honeypot and how well it can detect nuanced behaviours. The final section explores AI vs AI scenarios where AI systems attacked SYNAPSE. This part evaluates how SYNAPSE responds to these AI-driven attacks and its ability to map them accurately to the MITRE ATT&CK framework using its SYNAPSE-to-MITRE extension. These results provide critical insights into AI's evolving role in conducting and defending against cyberattacks, highlighting SYNAPSE's real-time ability to detect, respond, and map complex adversarial activities.

4.1 Experiment One: "Unmasking the Attackers"

The primary goal of this experiment was to understand how SYNAPSE could capture data from real-world cyber attackers. To achieve this, SYNAPSE was deployed in an open, exposed state on an Oracle Cloud Infrastructure (OCI) virtual machine (VM). Attackers were able to connect to the VM via two main methods: the first involved scanning known IP ranges of the Oracle Cloud Infrastructure, and the second involved accessing a deliberately leaked public IP address that was shared across platforms like "Pastebin"[1]. To facilitate access, a common and easily guessable username and password combination like *oracle/oracle* was set, ensuring that attackers could effortlessly log into the system. Upon gaining access, they generally did not engage with the Linux terminal presented by SYNAPSE. Instead, their behaviour suggested they were trying to conduct other malicious activities, such as using the machine as part of botnets, launching further attacks, or exploiting it for other illicit purposes. In this context, attackers might have been leveraging the machine for Distributed Denial of Service (DDoS) attacks, hosting malicious content, or scanning other networks from within the compromised system. Although they did not interact with the terminal as expected, this experiment demonstrated SYNAPSE's capability to extract critical data whenever a connection occurred. This allowed for a detailed analysis of global attack patterns, even when direct interaction with the system was minimal. The focus

[1] https://pastebin.com/

of this experiment, therefore, shifted to studying the connection metadata and understanding how SYNAPSE can gather and analyse this information to improve detection and response to malicious activities.

4.1.1 Securing the Trap

Several security measures were implemented on the VM hosting the honeypot to ensure that SYNAPSE remained a controlled environment and was not exploited to launch further attacks. First, "msmtp"[2], a lightweight SMTP client used for sending emails, was configured on the VM to alert the system administrator each time someone logged in or out of the machine. This was achieved by leveraging the Pluggable Authentication Modules (PAM) system, allowing scripts to be executed upon specific authentication events. In this case, a script was triggered during login or logout, sending an email containing critical information such as the user's or attacker's IP address and relevant metadata. This ensured real-time monitoring of any suspicious access to the machine.

In addition, how SYNAPSE is coded and executed on the VM inherently provides a layer of security. Attackers who connected via SSH were immediately confined within a Python script. If they attempted to close or bypass the script, their connection would be terminated, preventing them from directly interacting with the operating system. However, this mechanism alone was not foolproof, as sophisticated attackers might still attempt to inject malicious packets or exploit vulnerabilities that could allow them to break free from this containment and gain access to the underlying filesystem. If they succeeded, attackers could hijack the VM for malicious activities, such as launching further attacks, or even gain access to sensitive API keys for OpenAI and VirusTotal, leading to potential financial and operational damage to the research. To mitigate this risk, "msmtp" was used to send alerts whenever specific files on the filesystem were accessed. While this did not prevent from gaining access, it acted as a detection mechanism, allowing for real-time notifications if someone breached the security boundary and accessed critical files. This solution had pros and cons: while it provided immediate awareness of a security breach, it did not stop it. It relied on reactive measures rather than preventative ones, allowing for post-incident response rather than preemptive protection. To enhance this detection capability, "auditctl"[3], a tool used to manage the Linux Audit framework, was also employed. It allowed for continuous file access monitoring, generating logs and alerts whenever specific sensitive files were touched. This provided an additional layer of scrutiny to ensure any unauthorised access to critical files was swiftly detected.

Finally, a crucial step was taken to restrict the VM's ability to send specific types of packets in egress, effectively isolating the machine from initiating outbound connections. Limiting outgoing network traffic ensured that even if an attacker managed to infiltrate the VM, they could not use it to launch subsequent attacks or cause harm to other systems. This protected external machines from being compromised and reduced the risk of legal or ethical issues arising from the VM being part of a broader attack network. These security measures were designed to safeguard SYNAPSE and the underlying VM, ensuring attackers could connect to and interact with the system. However, they could not exploit it for further malicious purposes.

4.1.2 Captured Data

This experiment was conducted over ten days, during which the SYNAPSE honeypot was left exposed online, attracting numerous connections from attackers throughout the day. During this time, many different IPs attempted to access the system, with some returning on multiple occasions across different days, while others made several connections in quick succession. Throughout the experiment, **33 distinct IP addresses** were connected to the honeypot. This activity allowed

[2] https://marlam.de/msmtp/

[3] https://linux.die.net/man/8/auditctl

the collection of invaluable data on these interactions, which is critical for analysing patterns, behaviours, and potential threats. This analysis will not disclose the IP addresses to avoid legal concerns and respect privacy. Instead, the focus will be summarising the extracted data associated with these connections.

IP Reputations

Throughout the experiment, SYNAPSE was accessed by IP addresses with various reputations, ranging from negative to zero. IPs with negative reputations have been flagged for past malicious activities, while IPs with zero reputations indicate that no prior harmful behaviour has been recorded. However, the presence of zero-reputation IPs does not necessarily guarantee their harmlessness. During the study, there were several instances where IPs with zero reputation were flagged as malicious or suspicious in their most recent analysis. This could suggest that the IP addresses were newly compromised or recently repurposed for malicious intent. The delayed update of reputation databases may allow such IPs to remain undetected for some time before their activities are flagged. These cases highlight the importance of continuously updating reputation databases and the challenge of relying solely on reputation as an indicator of threat potential.

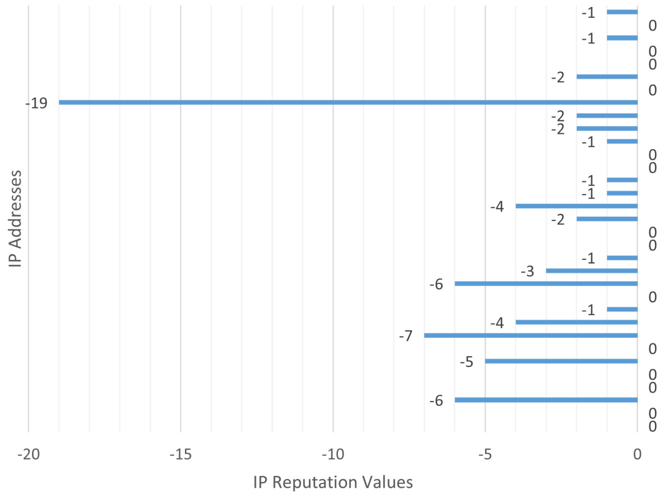


Figure 4.1. IP Reputations in Experiment One.

Graph 4.1 presents the reputation scores of individual IP addresses connected to the SYNAPSE honeypot during the experiment. Each horizontal line on the y-axis represents a single IP address, and the x-axis shows the corresponding reputation value. Reputation scores are negative values, with more negative numbers indicating a higher likelihood of the IP being associated with malicious or suspicious activity. For example, an IP with a reputation score of -19 represents one highly malicious and has a history of involvement in cyberattacks or harmful behaviour. On the other hand, IPs with scores closer to zero have a lower risk or may only be mildly suspicious, possibly flagged for unusual behaviour but not yet involved in significant attacks. In this graph, we can observe several IPs with reputation scores between -1 and -2, indicating lower levels of risk, and some IPs with more negative scores, such as -6, -7, and -19, showing the honeypot attracted a mix of highly suspicious and confirmed malicious actors.

Geolocations

During the experiment, a wide range of geolocation data was extracted from the IP addresses that connected to SYNAPSE. This data included information about the countries, regions, cities, time zones, and continents where the attackers were based. Extracting geolocation data is highly valuable, as it helps reveal patterns in the geographic origins of cyberattacks. Understanding where attackers are coming from can aid in attributing threats to specific regions, identifying hotspots of malicious activity, and shaping defence strategies accordingly. It also highlights the global nature of cyber threats, providing deeper insights into the tactics and behaviours of attackers worldwide.

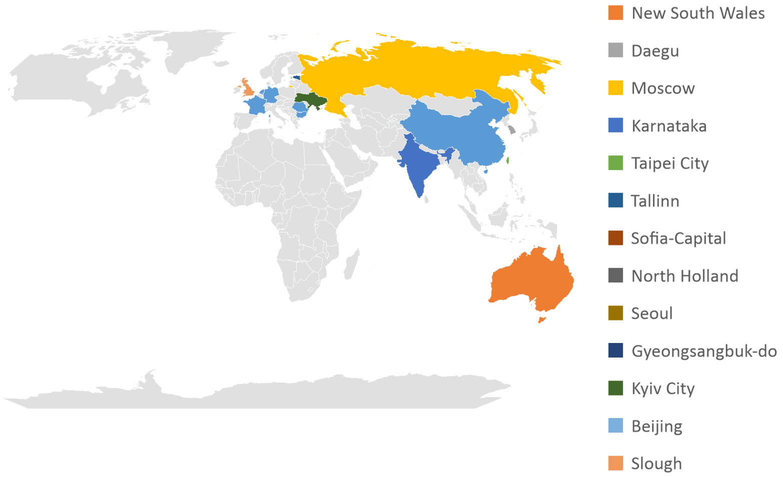


Figure 4.2. Geolocations in Experiment One.

Map 4.2 represents the geolocation information extracted from IP addresses connected to the SYNAPSE honeypot during the experiment. Each coloured region on the map corresponds to a specific city or region from which attackers accessed the system. The legend on the right side indicates the locations associated with these attacks, including significant areas such as Moscow, New South Wales, Karnataka, and Taipei City. In the context of the experiment, this map highlights the global nature of the attacks, demonstrating that SYNAPSE attracted connections from multiple continents, including Europe, Asia, and Oceania. The geolocation data extracted highlights an exciting pattern: most connections during the experiment came from eastern regions. For instance, areas like Moscow, Karnataka, and Taipei City are critical sources of malicious activity. This aligns with global trends where certain areas of the eastern hemisphere, especially Eastern Europe and parts of Asia, have been linked to higher volumes of cyberattacks. The prominence of Moscow in this data, the connection having -19 of reputation, could reflect the broader use of sophisticated cybercriminal infrastructure in that region, often associated with organised cybercrime or even state-sponsored operations. Additionally, the active presence of attackers from regions such as Taipei City, Daegu, and Karnataka suggests that these Asian areas are becoming increasingly essential hubs for cyber operations. This may be driven by the rapid growth of digital infrastructure in these regions, alongside rising technological expertise, which allows for both innovation and exploitation in cyberattacks. The fact that most connections came from the eastern areas also ties into time zone patterns. Many attacks are likely timed to coincide with peak local working hours, reflecting an organised approach from attackers who may be operating when defenders in other parts of the world are less active. This global time zone dynamic could explain why certain regions are more prominent in the data, as attackers exploit not only geographic vulnerabilities but also the rhythms of global internet activity. In conclusion, the accuracy of these geolocation results varied, with SYNAPSE achieving impressive precision in some cases, identifying locations within a 10 km range of the actual source. However, the accuracy was broader in other instances, with up to 1000 km deviations. Despite these variations, the data remains instrumental in mapping the regions from which the attacks originated.

Number and Duration of Connections

The number of connections and their durations, extracted by SYNAPSE for each IP address, are essential metrics in this experiment. The number of connections helps identify persistent attackers, with repeated attempts potentially indicating a more determined or automated effort. Connection durations reveal the nature of the interactions, where longer durations suggest active engagement with the system, while shorter ones may indicate quick scans or automated attempts. Together, these metrics offer insights into attacker behaviour, persistence, and potential risks, enhancing our understanding of the threats targeting the honeypot.

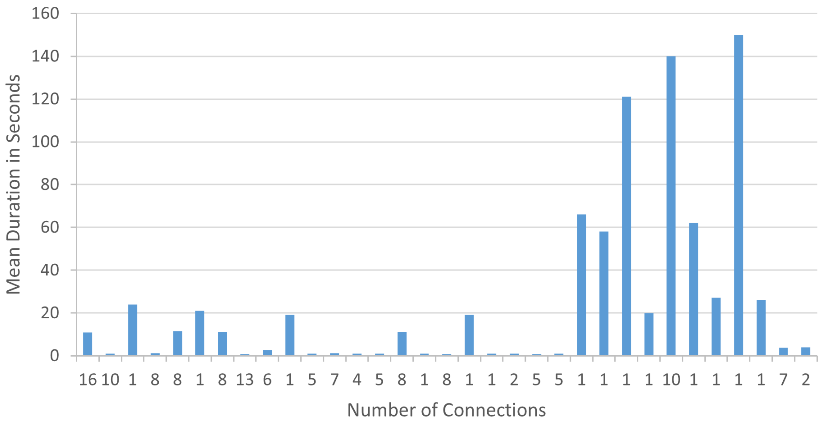


Figure 4.3. Number of Connections with relative Durations in Experiment One.

Graph 4.3 illustrates the relationship between the number of connections made by single attackers and the corresponding mean duration of these connections in seconds. It is possible to observe a wide variability in the duration of connections, with some attackers maintaining very brief interactions while others remained connected for significantly more extended periods. Most connections with lower counts, typically between one and eight, are characterised by short durations. This suggests that many attackers were likely performing quick, automated activities such as scanning or probing the system for vulnerabilities without prolonged interaction. However, there are several notable instances where attackers with just a single connection stayed connected for much longer, up to around 150 seconds. This suggests more deliberate interactions, where attackers may have attempted more sophisticated actions requiring a more extended connection. On the other hand, there are cases where attackers made multiple connections, up to 16, but their average connection durations remained relatively short. This could indicate repeated automated attempts to breach the system or exploit different vulnerabilities without staying connected for extended periods. These attackers may have been using scripts or bots designed to rapidly attempt a wide range of attacks, possibly hoping to find a weakness without needing sustained interaction. Overall, the data extracted by this experiment reveals two key behaviours: attackers who engage in quick, repeated connections that are likely automated and those who maintain longer, singular connections, suggesting more engaged, possibly manual attempts to explore the system.

Time of Connection

Understanding the time intervals during which connections to the SYNAPSE honeypot occurred is crucial for identifying patterns in attacker behaviour. Time interval correlation can help reveal peak activity periods, showing when attackers are most likely to engage with the system. By analysing these intervals, we can determine whether attacks are concentrated around specific times. This might indicate automated attacks triggered at regular intervals or time zone-related patterns, where attackers are active during local working hours. Additionally, time interval analysis can uncover whether specific attacks are coordinated or sequential, helping to differentiate between random, opportunistic threats and more organised, persistent attempts.

Graph 4.4 displays the number of connections to the SYNAPSE honeypot across four distinct time intervals, highlighting variations in attack activity throughout the day. The first time interval, from approximately 00:14:10 to 06:10:13, shows the highest number of connections, with 25 recorded during this period. This suggests that many attacks occurred during the early morning hours, which may indicate that many connections were automated, likely running during off-peak hours to avoid detection or take advantage of quiet periods on the network. In the second time

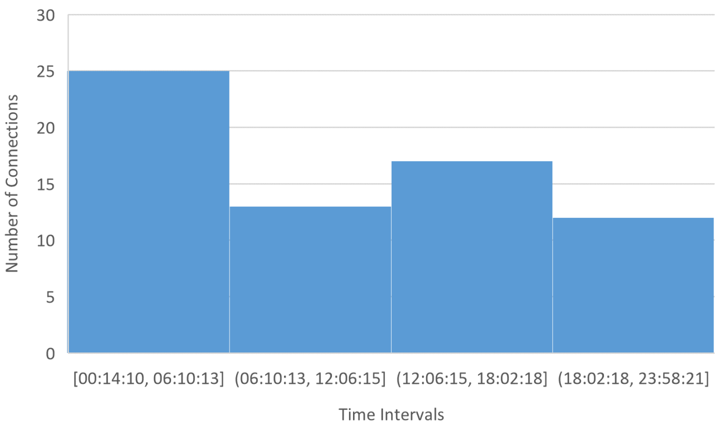


Figure 4.4. Time Intervals and relative Number of Connections in Experiment One.

interval, from 06:10:13 to 12:06:15, the number of connections drops significantly, with only 12 connections recorded. This decline in activity could suggest a temporary lull, possibly due to the attacks' automated nature or the targeting of different time zones where attackers might not be as active during this period. Interestingly, in the third interval, between 12:06:15 and 18:02:18, there is a resurgence in connections, with 17 recorded during this period. This midday increase may suggest that the attackers, or their automated systems, are programmed to strike during certain times of the day when network defences might be weaker. It could also indicate activity timed to match specific time zones, where attackers are more active during business hours or early afternoon. Finally, during the last interval, from 18:02:18 to 23:58:21, the number of connections drops again, down to 11. This pattern likely reflects a winding down of activity towards the end of the day, further suggesting that many of the connections may have been automated, scheduled to operate during specific time windows, and that there were fewer manual attacks during these late hours.

4.2 Experiment Two: "Through the Eyes of the Intruder"

This section presents the results of the second experiment. The primary objective was to evaluate SYNAPSE's ability to enhance cybersecurity by observing user interactions without providing any information about the systems they were engaging with. **18 individuals**, with varying levels of expertise, were tasked with interacting with two separate machines, one running SYNAPSE and the other running DENDRITE. Crucially, the participants were not told that these systems were honeypots nor that one incorporated generative AI. This deliberate lack of context ensured that their interactions would be natural and free from bias, providing a more precise test of how effectively SYNAPSE could deceive users. Participants were free to engage with the machines however they saw fit, whether through essential exploration or attempting direct attacks on the system. Some opted to combine both approaches. The absence of specific instructions or context was a critical design choice, ensuring their actions and decisions were uninfluenced by preconceived notions. Each participant was fully informed that their IP addresses and geolocations would be tracked during the experiment, and they provided consent for this data collection before proceeding. Following the interaction, participants were asked to complete a comprehensive survey to capture their experiences. Key questions focused on whether they detected anything unusual or artificial about the systems and if they could discern that one of the environments was not entirely natural. This feedback was essential to evaluating SYNAPSE's ability to simulate a real-world system convincingly while operating in parallel with DENDRITE.

This section will detail the impressive results of these interactions, exploring how SYNAPSE's AI capabilities enhanced the security and deception mechanisms and demonstrated its superiority over a traditional honeypot like DENDRITE. By analysing user behaviour, it will be possible to understand how well SYNAPSE disguises itself and misleads intruders, providing strong evidence of its potential in real-world cybersecurity applications.

4.2.1 Interaction Types

The interactions between participants and the systems were categorised into three main types: basic interaction, attacking the system, and combining both. Participants were free to interact with the machines in any way they preferred, which provided a diverse range of behaviours for analysis.

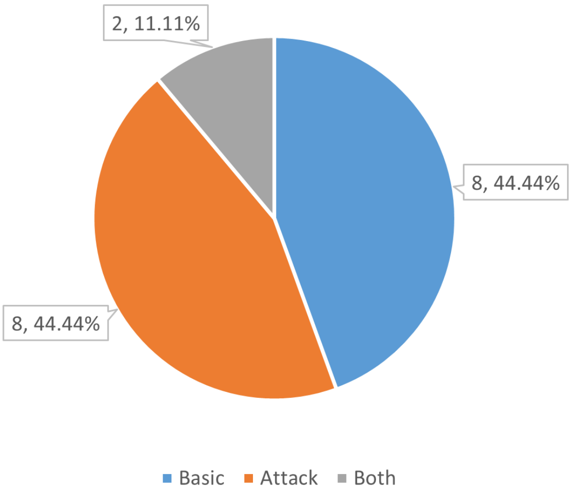


Figure 4.5. Interaction Types in Experiment Two.

Basic interaction involved navigating the file system, creating or modifying files, and other standard, non-aggressive activities. As shown in Figure 4.5, 44% of participants engaged solely in these benign interactions, focusing on exploration without attempting to compromise the system. This group approached the machines like a regular system, providing insights into how well the honeypots simulated an authentic environment for standard users. Conversely, 44% of participants took a more aggressive approach by attempting to attack the systems. This group tested the machines for vulnerabilities or tried to launch disruptive actions, simulating a direct assault on the system's security. A smaller subset of participants, comprising 11%, combined both approaches. These users initially engaged in basic exploration but later transitioned to attacking the system, reflecting a scenario where one might first explore the architecture before launching an offensive.

The experiment also tested SYNAPSE's AI capabilities to determine whether an interaction constituted an attack. The AI performed exceptionally well, distinguishing between benign interactions and attacks with perfect accuracy. After each session, the AI's classification of whether an attack occurred was compared to the participants' self-reported interaction types (basic, attack, both). In every case, the AI's evaluation aligned perfectly with the participants' declarations, demonstrating the system's robust ability to detect malicious behaviours with high precision.

4.2.2 Participants Perception of Systems Behavior

Following the experiment, survey questions were administered to assess how participants perceived the systems they interacted with. These questions aimed to gauge whether the users noticed any differences between the two machines, detected any unusual behaviour or felt that one or both were inaccurate.

Perceived Similarity in System Interaction

Participants were asked whether they found the two systems to be similar regarding interaction. This question was designed to determine how well SYNAPSE, with its AI-driven adaptability, blended with the static equivalent DENDRITE from the user experience perspective. If participants perceived the two systems as identical, it would suggest that SYNAPSE successfully mimicked a traditional environment without revealing its AI-enhanced capabilities. On the other hand, if participants noticed differences, their responses could offer insights into whether SYNAPSE's dynamic nature or DENDRITE's static characteristics were more apparent during the interaction. Answers to the question "Did the two machines seem identical to you in terms of interaction?" are reported in Table 4.1, where the "first" machine indicates SYNAPSE, while the "second" one is DENDRITE.

Participant	Answer
1, 3, 5, 11, 14, 16, 17	"Yes."
2	"No, the second one had longer response times."
4	"No, the second machine seemed more robust. For example, it didn't allow the execution of *rm -rf /tmp*, while the first one did."
6	"No, the first machine had basic Linux functionalities as executable, whereas the second was much more restrictive with user permissions."
7	"No, they did not seem identical to me as the response times differed."
8	"No, they offered a different set of binaries in the */bin* directory, with a much wider selection on the second machine."
9	"No, I noticed that the second machine had an interface under the terminal that was not visible on the first machine."
10	"No, the second machine requires precise commands, while the first seems to understand and tolerate minor errors."
12	"No, for example, the second machine did not have *sudo*."
13	"No, there were some differences between the two. For example, one closed the connection when attempting to run *sudo*, while the other stated that the command wasn't available. They had different permissions regarding the database, and when using unavailable commands like *vim*, they gave different responses."
15	"No, the second machine felt more restricted, offering fewer possibilities for movement. When trying commands like *ifconfig*, *open*, or *arp*, it returned *command not found*, whereas the first machine behaved differently. For instance, it prompted to install *ifconfig*, denied access to *open*, and successfully executed *arp*. This made the first machine seem more realistic overall."
18	"No, the first one was slightly slower and didn't allow me to delete anything."

Table 4.1. Answers to First Survey Question in Experiment Two.

Seven participants noticed nothing unusual during their interactions with the two machines. However, three participants commented on the different response times between the systems. This disparity is unavoidable, as SYNAPSE requires time for the AI to generate responses. To mitigate this, the DENDRITE machine was placed in the United States while all participants were in Europe. This setup aimed to create a comparable delay in response time for both systems, ensuring the experience felt as similar as possible despite the underlying technical differences. Several participants observed more distinct differences between the machines. One described the second one as feeling more robust, while another found it much more restrictive regarding user permissions. Others pointed out variations in the set of binaries available and noted different responses to commands like *sudo*. These observations are intrinsic to the experiment's design. On the one hand, SYNAPSE's terminal is simulated using AI, meaning many functionalities are handled directly by the AI rather than being explicitly configured by a system administrator. In contrast, the second system, being a traditional setup, required detailed manual configuration. The discrepancies participants noticed reflect that while SYNAPSE can automate many aspects of system management via AI, exact parity with a manually configured system can be challenging. This also highlights an essential advantage of using AI-driven systems like SYNAPSE: configuration times can be significantly reduced, as the AI automatically manages many functionalities. However, there is a trade-off, as indicated by another participant's comment that the first machine allowed the execution of commands even when they were incorrect. This could be attributed to AI "hallucinations" or instances where it fails to adhere strictly to system rules. Such errors underscore the potential for inaccuracies in AI-driven environments, meaning that while it can simplify system management, it does not always guarantee 100% rigorous functionality. This issue will be explored in more detail later in the analysis.

Detection of Anomalous Responses

Participants were asked whether they noticed anomalous responses from either terminal during their interactions. This question assessed whether SYNAPSE generated any unusual or unexpected replies and, more importantly, whether participants could detect these anomalies. Understanding how well the AI's responses aligned with user expectations is crucial for evaluating SYNAPSE's ability to simulate a natural system convincingly without revealing its AI-driven nature. Answers to the question "Did you notice any anomalous responses from either of the two terminals?" are reported in Table 4.2, where the "first" terminal indicates SYNAPSE. In contrast, the "second" one is DENDRITE.

Participant	Answer
1, 5	"Yes, the second terminal had coloured text, while the first did not."
2, 3, 6, 7, 8, 9, 12, 14, 15, 16, 17, 18	"No."
4	"Yes, in the second terminal *sudo* is not recognized as a command."
10	"Yes, the second machine requires precise commands, while the first seems to understand and tolerate minor errors."
11	"Yes, when I tried to update or delete commands in the databases of the first machine, I received a response indicating that the administrator blocked the operation."
13	"Yes, for example, the first terminal gave me *Permission Denied* when I tried to use *vim* to open a file without special permissions. I also got a strange response when I attempted to delete a database. In the second terminal, I noticed odd behaviour as well. For instance, when I made a typo, the response was unusual: I mistakenly typed *mysql 'u enea 'p* instead of *mysql -u enea -p*, and it returned *Access Denied.*"

Table 4.2. Answers to Second Survey Question in Experiment Two.

Of the 18 participants, **12 did not notice abnormal responses** from the terminals. This is a promising outcome, indicating that SYNAPSE's AI was able to generate responses that did not raise any suspicion about the system's authenticity. Since participants had no prior context about the experiment, their lack of concern suggests that they interacted with the terminal as if it were a natural system without doubting its legitimacy. For the remaining responses, beyond the observations made for the previous question regarding system differences, a few participants did experience issues that led them to question the reality of the system. For one user, commands worked even when they were incorrect, and for another, the MySQL service returned a strange output. These instances highlight moments where the AI failed to generate an accurate or appropriate response, leading participants to sense something was off. These errors represent lapses in the AI's ability to simulate a legitimate system perfectly and are a reminder that AI-driven terminals, while powerful, can still produce computational mistakes. Finally, two participants mentioned that the terminal in DENDRITE displayed graphical colours, unlike SYNAPSE. This is a configuration difference that was not accounted for in this experiment. SYNAPSE lacks graphical enhancements like coloured terminal text, which could be considered a potential future improvement to make the AI-driven system even more convincing and aligned with real-world environments.

Perception of System Non-Authenticity

Participants were ultimately asked whether they felt they were interacting with a non-real system. This question aimed to assess if, after their experience, they doubted the authenticity of one or both machines and concluded that the environment might not be genuine. Detecting any signs of artificiality would suggest the system failed to deceive the user entirely, while a lack of doubt would indicate that SYNAPSE successfully mimicked a real-world system. Answers to the question "Did it seem to you that you were interacting with a non-real system?" are reported in Table 4.3, where the "first" terminal indicates SYNAPSE. In contrast, the "second" one is DENDRITE.

Participant	Answer
1, 2, 3, 5, 7, 8, 9, 14, 16, 18	"No."
4	"Yes, the second one is a bit strange."
6	"Yes, on the second one, because many basic Linux executables were missing, which makes it very difficult to configure a real system."
10, 11	"Yes, the first one."
12	"Yes, the second one."
13	"Yes, some responses in the first were strange. It didn't let me download anything, transfer files between the machines, or use sudo. The responses were generally slightly odd, especially when it couldn't handle a task."
15	"Yes, the first system felt more real. It provided different responses to different commands depending on the input. As mentioned earlier, it suggested installing them using root privileges even for unrecognised commands. On the other hand, the second system always returned the same response: *command not found*."
17	"Yes, because of the commands, it felt like Linux and Unix."

Table 4.3. Answers to Third Survey Question in Experiment Two.

The results of the third question revealed that **10 participants did not feel like they interacted with a non-real system**. This positive outcome indicates that SYNAPSE performed well in mimicking a natural OS. The fact that most users did not suspect anything artificial about their experience suggests that SYNAPSE could convincingly replicate a traditional environment, fulfilling its primary function as a honeypot designed to deceive potential intruders. However, three participants noticed that the first system was inaccurate, citing strange or incorrect responses during the interaction. These instances highlight areas where SYNAPSE's AI-generated responses failed to fully maintain the illusion of authenticity, offering valuable feedback for future improvements.

The remaining participants indicated that they believed the second machine was fake; some even provided detailed explanations for their reasoning. This is significant because it means that they did not suspect the first system, driven by AI, of being artificial. Instead, they focused on the second, real machine, showing how well SYNAPSE could disguise its AI integration. One participant was led to this conclusion because the second machine lacked many executable commands, which, as explained earlier, reflects that the traditional system requires more manual configuration and, thus, more time to set up fully. Another user felt the responses from the first terminal seemed more natural and ordinary than the second, further reinforcing SYNAPSE's ability to generate convincing interactions. In one notable case, a participant did not realise at all that one of the systems was powered by AI. Instead, they assumed the difference was that one system was Linux and the other was Unix, highlighting how effectively SYNAPSE concealed its AI-driven nature.

4.2.3 Identifying the Real System

After gathering feedback, a key question was posed to underline the most important outcome of the experiment, focusing on the comparison between SYNAPSE and DENDRITE. Participants were asked, "If you had to choose one of the two systems, in terms of similarity or resemblance to a real one, which one would you choose?". This question aimed to determine which system felt more authentic to the participants. The goal was to understand whether SYNAPSE's AI-driven responses were compelling enough to make it indistinguishable from a traditional architecture like DENDRITE. The outcome of this question would highlight which system participants perceived as more realistic, serving as a critical measure of SYNAPSE's ability to deceive and blend into a genuine computing environment. This result is crucial for evaluating SYNAPSE's performance, as it directly reflects its success in simulating a natural system better than a traditional, manually configured honeypot like DENDRITE.

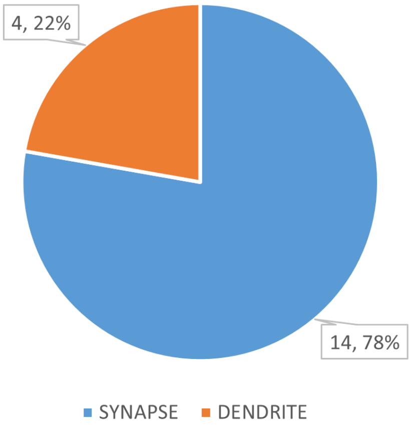


Figure 4.6. SYNAPSE vs DENDRITE in Experiment Two.

The results of this crucial question are astonishing and very promising. As shown in Figure 4.6, **almost 80% of the participants** (14 out of 18) **chose SYNAPSE as the more real system**, despite it being entirely simulated by AI. This outcome demonstrates the efficiency of SYNAPSE not only in mimicking a real machine but also in disguising its artificial nature to the point where it was perceived as more authentic than DENDRITE, which is a genuine system. This marks a significant achievement in the ability of SYNAPSE to create an environment that can effectively deceive users. Among the four users who voted DENDRITE as the natural system, one cited SYNAPSE's longer response times as their choice. However, the remaining three based their decision on noticeable errors or "hallucinations" from SYNAPSE. These errors were significant enough that they couldn't be overlooked, such as allowing incorrect commands to be executed, something that should never happen in a natural system. These responses highlight that while SYNAPSE's AI generally performed well, some noticeable errors remained. In most cases, however, SYNAPSE either made no errors or exhibited minor inaccuracies that were not obvious enough to raise suspicion, especially for participants who had no context and did not know they were interacting with an AI-driven machine. It is also important to note that among the participants who selected SYNAPSE as the natural system, four were highly experienced, including computer engineers with expertise in cybersecurity. This underscores SYNAPSE's ability to deceive average or intermediate users of a Linux terminal and those who regularly work with such systems and possess high expertise. This further emphasises SYNAPSE's potential in cybersecurity applications, demonstrating that it can convincingly trick even seasoned professionals.

It is worth noting that some participants chose SYNAPSE over DENDRITE due to the latter's more restricted configurations. However, this is not necessarily a negative outcome. It further underscores SYNAPSE's ability to disguise its AI nature convincingly, making it appear more accurate and accessible than the manually configured DENDRITE. This highlights a key advantage of SYNAPSE: it is easier to configure and requires less setup time than a traditional system. The fact that, with less configuration effort, it could surpass a genuine machine in perceived authenticity is a significant strength, demonstrating its efficiency in simulating real-world environments and reducing administrative overhead.

Perceived Temptation

In the final stage of the experiment, participants were asked one last key question to determine which system they found more tempting from the perspective of an attacker: "If you were an attacker, which of the two systems would seem more 'tempting' (where you could obtain more results from executing an attack or extract more valuable information)?". This question was crucial in assessing not just the realism of the systems but also their ability to attract potential attackers. A machine that appears vulnerable or more open to exploitation will likely draw more attention from malicious actors. If SYNAPSE could convincingly present itself as a more tempting target than DENDRITE, it would further demonstrate its effectiveness as a honeypot. The more convincing and attractive SYNAPSE appears, the greater its value for capturing and analysing malicious behaviours.

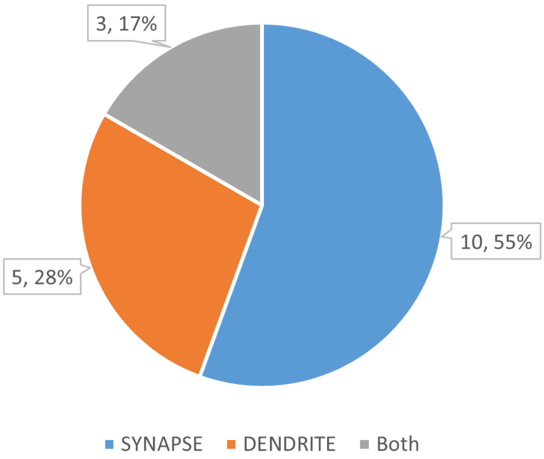


Figure 4.7. Perceived Temptation in Experiment Two.

The results of the final question (see Figure 4.7) reveal a clear trend: **the majority of participants** (55%) **found SYNAPSE** to be **the more tempting system** from an attacker's perspective. This is a significant outcome, as it suggests that SYNAPSE's dynamic generation of content, such as system responses and services like MySQL, is highly effective in creating an environment rich with potential exploits or valuable information. SYNAPSE's ability to simulate a dynamic and complex system is more appealing to attackers than DENDRITE, where the content is static and manually configured. In contrast, 28% of participants chose DENDRITE as the more tempting system. This could reflect that a more static machine might seem familiar and potentially easier to exploit for certain types of attackers, especially those looking for predictable configurations or common vulnerabilities. However, this group was a minority, indicating that SYNAPSE's AI-driven responses made it appear a more complex and exciting target for most participants. Interestingly, 17% of participants found both systems equally tempting, which may suggest that while SYNAPSE's AI can dynamically adapt and create convincing environments, some participants felt that both systems offered valuable opportunities from an attacker's point of view. Overall, these results underline the strength of SYNAPSE's AI in simulating a convincing real-world system and creating an environment that appears highly appealing to potential attackers. The dynamic nature of SYNAPSE, capable of generating responses and interactions on the fly, outperforms the more rigid, static structure of DENDRITE in terms of attractiveness for exploitation.

Notable Participant Interactions

Among the most notable interactions, two participants with high levels of expertise spent more than an hour attempting to break SYNAPSE through privilege escalation attacks. Analysing the logs of their sessions revealed that they tried many methods available to them, repeatedly connecting to the machine for long, continuous sessions. Despite their efforts, these users remained entirely unaware that they were interacting with a simulated environment, wholly convinced that they were dealing with a natural operating system. This is an extraordinary result, showcasing the impressive capabilities of SYNAPSE to trick even expert users. These individuals, familiar with Linux systems and experienced in cybersecurity practices, could not detect that the environment was artificial. SYNAPSE's ability to generate dynamic, convincing responses to their repeated privilege escalation attempts ensured they never questioned the system's legitimacy. One of the most promising aspects of this outcome is that SYNAPSE, as an AI-driven simulation, could mimic a natural environment so effectively that these advanced users persisted in their efforts for an extended period. Even though there is no natural operating system, just a Python script emulating one, SYNAPSE successfully concealed this by responding in ways that maintained the illusion of a functioning OS. There was no possibility for privilege escalation to succeed because there was no actual system to exploit; it was all an AI-generated facade.

4.3 Experiment Three: "AI as the Adversary"

In this final experiment, we shift the traditional roles in cybersecurity by positioning AI as the active adversary. Unlike the previous evaluation cases, this one utilises an advanced generative AI model, specifically "gpt-4o", to execute a series of targeted cyberattacks. The primary objective of this experiment is to measure how well SYNAPSE can automatically interpret and map the generated attack logs to specific techniques within the MITRE ATT&CK framework, leveraging its SYNAPSE-to-MITRE extension. Three potential mappings are expected to emerge from the attack logs, providing a comprehensive view of how well SYNAPSE identifies, processes, and aligns the detected behaviours with known adversarial techniques in MITRE ATT&CK. This experiment will help validate the automated attack detection system's effectiveness, highlighting areas where improvements may be necessary. The concept of "AI as the Adversary" challenges traditional cybersecurity methodologies by introducing an adaptive, learning-based threat that evolves with the environment.

4.3.1 Pre-Selected MITRE ATT&CK Techniques

In this experiment, the **AI will perform 40 targeted attacks against SYNAPSE**. Unlike the standard case study explained in Section 3.4.2, where the attacks were randomly generated, this experiment adopts a more structured approach. Specifically, 40 distinct MITRE ATT&CK techniques, selected to span the entire Enterprise Matrix, have been chosen as the attack vectors. The rationale for pre-selecting the attack techniques is twofold. First, it guarantees that the SYNAPSE-to-MITRE extension is tested against various tactics and techniques, thoroughly assessing its mapping capabilities across the complete MITRE ATT&CK database. Second, as discussed in later sections, calculating performance metrics for the model's classification requires building a dataset where the correct mappings are known. By asking the AI to execute attacks aligned with specific MITRE techniques, the need to map post-execution manually is eliminated. This method helps reduce errors in selecting the correct mappings, ensuring that the performance and metrics calculations are accurate and reliable.

4.3.2 Metrics Calculation on Real Data

Once all 40 attacks from the AI have been executed, the SYNAPSE-to-MITRE extension is activated to map each one to three possible MITRE ATT&CK techniques. A mapping is considered positive if at least one of the three aligns with the correct technique. As previously mentioned, the model achieved an F1-score of 89% during training; however, this was calculated on training data. We expect this percentage to decrease when applied to real-world scenarios, as the model is now tested on new data it has not encountered before, providing a more realistic evaluation of its performance.

A new dataset calculates metrics, containing one row for each of the 40 AI-performed attacks. It mirrors the structure used to train the SYNAPSE-to-MITRE model but is specifically tailored for metric calculation over these new, unseen attacks. Each row includes the following key fields: the correct technique label, sub-technique, and technique name as selected by the MITRE ATT&CK Matrix. These precise labels are provided at the outset, as the attacks were selected based on predefined MITRE techniques. Additionally, each row contains an unstructured Cyber Threat Intelligence (CTI) sentence, which the SYNAPSE-to-MITRE AI generates during mapping. This sentence summarises the actions taken during the attack and is unique to each log entry. A portion of the dataset used for this experiment can be seen in Table 4.4.

The real-world dataset allows us to systematically evaluate the model's performance by comparing its predictions with the known correct mappings. The generated sentence is passed to the SYNAPSE-to-MITRE model for each attack log, which outputs three potential mappings. The model then checks if any of these correspond to the correct technique labels and the name in the same row. This process enables the calculation of three key metrics: precision, recall, and F1-score. Precision measures the proportion of correct mappings among those predicted, recall assesses the proportion of correctly identified techniques relative to the total number of proper techniques, and the F1-score provides a harmonic mean of precision and recall, offering a balanced view of the model's performance.

4.3.3 SYNAPSE-to-MITRE Performance

The results of the experiment show that the SYNAPSE-to-MITRE extension achieved a precision of 0.75, a recall of 0.68, and an F1-score of 0.70. These metrics provide a comprehensive view of how well the system performs when subjected to real-world data, highlighting the balance between precision and recall in the context of automated attack mapping.

Precision	**Recall**	**F1-Score**
75%	68%	70%

Table 4.5. Metrics in Experiment Three.

The **precision** score of 0.75 indicates that **75% of the mappings** generated by SYNAPSE-to-MITRE **were correct**. This demonstrates that, in most cases, when the system predicts a particular MITRE ATT&CK technique, the prediction aligns with the actual attack behaviour. High precision is critical in minimising false positives, as it ensures that most of the methods identified by the system are valid. A precision value like this is encouraging, showing that the system is relatively confident in its mappings and that false positives are not overwhelming the process. However, precision alone is insufficient to understand the full scope of the model's performance. This brings us to **recall**, a crucial metric for evaluating the model's ability to

Attack	Label Technique	Label Sub-Technique	Technique Name	Sentence
0	T1548	T1548	Abuse Elevation Control Mechanism	Adversaries may exploit setuid and setgid permissions to escalate privileges by creating and executing a custom binary.
1	T1098	T1098	Account Manipulation	Adversaries may inject malicious SSH keys into the authorized_keys file to gain unauthorized access to systems.
2	T1543	T1543	Create or Modify System Process	Adversaries may attempt to create or modify system services to establish persistence on a compromised system.
3	T1053	T1053	Scheduled Task/Job	Adversaries may attempt to escalate privileges by inserting malicious cron jobs into a MySQL database to modify the sudoers file.
4	T1059	T1059	Command and Scripting Interpreter	Adversaries may create and execute malicious scripts to perform unauthorised actions on a system.
5	T1653	T1653	Power Settings	Adversaries may attempt to persist by creating malicious events or scripts that execute at system reboot.
6	T1059	T1059	Command and Scripting Interpreter	Adversaries may modify user configuration files to execute commands automatically upon user login.
7	T1222	T1222	File and Directory Permissions Modification	Adversaries may attempt to change file permissions to gain unauthorised access or control over files and directories.
8	T1564	T1564	Hide Artifacts	Adversaries may attempt to make files difficult to discover by hiding them in directories and modifying system configurations to avoid detection.
9	T1562	T1562	Impair Defenses	Adversaries may disable or stop security tools to evade detection and hinder defences.
10	T1562	T1562	Impair Defenses	Adversaries may delete or modify artefacts generated within systems to remove evidence of their presence or hinder defences.

Table 4.4. Dataset Fragment in Experiment Three.

detect all relevant attacks. With a recall of 0.68, **the model** was able to correctly **identify about 68% of the actual attack techniques**. This score is slightly lower than precision, indicating that while SYNAPSE-to-MITRE is good at being accurate when it makes a prediction, it may miss some attacks. A lower recall suggests that there are instances where the system fails to detect specific techniques, leading to false negatives. In real-world scenarios, this means that some attack behaviours might slip through without being adequately mapped, which could be a limitation in identifying the full scope of adversarial activity. The balance between precision and recall is summarised by the **F1-score**, which in this case is **70%**. It represents the harmonic mean of precision and recall, offering a single metric that balances both. In this context, the F1-score suggests that while the system performs reasonably well in precision and recall, there is room for improvement in achieving a more consistent detection rate. The score being closer to precision reflects that the system prioritises correctness over completeness, which can be a strategic trade-off depending on the intended use.

When discussing the relationship between these metrics, it's essential to recognise the inherent trade-off between precision and recall. High precision ensures fewer false positives, but achieving this often means the model is more conservative in its predictions. This leads to lower recall because some attack behaviours might not be detected. In contrast, if recall were to be improved by making the model more aggressive in its predictions, precision might drop as the system starts flagging more false positives. The balance achieved here with an F1-score of 0.70 suggests that SYNAPSE-to-MITRE has been tuned to maintain a relatively high level of accuracy while still capturing the majority of attack techniques, but further optimisation could be explored to enhance its recall without significantly sacrificing precision.

Mapping Errors and Improvement Opportunities

To further illustrate the results, let's consider a specific example. One attack, which should have been correctly mapped to *Power Settings [T1653]*, was instead inaccurately mapped by the SYNAPSE-to-MITRE system. The three mappings generated were *Event Triggered Execution [T1546]*, *Command and Scripting Interpreter [T1059]*, and *Boot or Logon Autostart Execution [T1547]*. The system generated the following sentence to describe the attack: "Adversaries may attempt to persist by creating malicious events or scripts that execute at system reboot".

In this case, the mappings produced were related to persistence mechanisms, but none aligned directly with *Power Settings [T1653]*. The sentence generated by SYNAPSE-to-MITRE, while informative, lacked specificity. It broadly suggested persistence techniques, but the language used failed to pinpoint the unique characteristics of the "Power Settings" attack. For example, the model interpreted the term "system reboot" and associated it with boot or logon persistence techniques, leading to inaccurate mappings. This generalisation could have caused the system to associate the attack with techniques that seemed more aligned with script execution and event triggering, which diverted the mapping away from the correct method.

This example sheds light on one of the potential causes behind the recall rate of 0.68. While the system demonstrates vital precision in cases where the correct mappings are present, situations like this highlight its limitations in fully capturing the nuances of certain attack behaviours. Although the generated sentence was relevant to persistence, it was not specific enough to guide the system toward the correct technique. This implies that one of the areas for improvement could be the refinement of the AI's sentence generation process, making it more detailed and aligned with the exact behaviours of specific techniques. Enhancing the precision of these descriptions could help improve recall by better directing the system toward the correct mappings in more complex or nuanced attacks. The F1-score of 0.70 reflects the balance between these factors. While SYNAPSE-to-MITRE demonstrates a solid ability to map many attacks correctly, this case underscores its challenges in capturing all relevant techniques. If the generated sentence had been more specific about the nature of the "Power Settings" manipulation, recall might have improved, potentially lifting the system's overall performance.

Chapter 5

Conclusions

The constantly evolving landscape of cybersecurity demands more sophisticated and adaptive defence strategies. This thesis explored the intersection of honeypot technology and generative AI to address the limitations of traditional static defences, proposing a dynamic system capable of evolving in real-time. This research's core is SYNAPSE (Synthetic AI Pot for Security Enhancement), an innovative AI-driven honeypot designed to simulate realistic environments and interact with attackers dynamically. By integrating machine learning technologies and automatic log mapping to the MITRE ATT&CK framework leveraging its SYNAPSE-to-MITRE extension, SYNAPSE significantly enhances the capability to detect, map, and respond to cyber threats in real-time. Throughout this work, experiments were conducted to evaluate the effectiveness of SYNAPSE compared to traditional honeypot systems. The findings demonstrate how combining AI with honeypots strengthens cybersecurity defences and shifts the paradigm from passive monitoring to active, anticipatory threat detection.

Chapter 1 introduced the thesis by examining the challenges of the modern cybersecurity landscape, where the evolving nature of threats renders static defence strategies insufficient. It discusses the Defender's Dilemma, highlighting the imbalance between attackers who exploit single vulnerabilities and defenders who must protect all entry points. The chapter argues for the necessity of a dynamic approach and sets the stage for SYNAPSE.

Chapter 2 comprehensively reviewed this thesis's foundational concepts. It discusses information security and cybersecurity, highlighting core principles, the evolving threat landscape, and emerging trends. The chapter then delves into the role of honeypots in cybersecurity, covering their objectives, classifications, ethical considerations, and real-world examples. The following sections explore generative artificial intelligence (AI), particularly large language models (LLMs) and prompt engineering, which are crucial to SYNAPSE's dynamic capabilities. The review also addresses machine learning (ML), focusing on types of models, training methods, and their application in natural language processing (NLP). Finally, the chapter discusses the MITRE ATT&CK framework, detailing its structure, advantages, and relevance to SYNAPSE's automatic log mapping. The literature review concludes by examining AI applications in cybersecurity, focusing on integrating honeypots and AI, comparing this approach with traditional honeypots, and identifying the advantages and limitations of generative AI-enhanced honeypot systems.

Chapter 3 outlined the methods used to develop and implement SYNAPSE. It is a Python-based system that simulates a Linux OS terminal and critical services such as SSH and MySQL servers, leveraging generative AI. The chapter begins by describing the classification and setup of SYNAPSE. Although primarily categorized as a low-interaction honeypot, it incorporates features typically found in higher-interaction systems, allowing attackers to engage more deeply while remaining in a controlled environment. SYNAPSE was deployed in a virtual environment, offering scalability and flexibility while optimizing resource use for better performance during simulations. The methodology section also outlines the development of SYNAPSE, tracing its evolution from the initial shelLM project to its current form. Its main features include prompt engineering techniques and large language models (LLMs) to simulate realistic interactions. How the AI processes requests is detailed, illustrating how it computes responses in real-time based on attacker behaviour. Additionally, SYNAPSE maintains a memory of user interactions, allowing multiple sessions for the same user to enhance the realism of the simulated environment. The system collects extensive data from attacker interactions, including command logs, IP addresses and reputations, ports used, geolocation information, and connection numbers and duration. This data is vital for analyzing and understanding attacker behaviour patterns. One of the most significant innovations in SYNAPSE's methodology is the SYNAPSE-to-MITRE extension, which automates the mapping of attack logs to the MITRE ATT&CK framework. This feature leverages machine learning techniques to quickly identify attack tactics and provide real-time insights for defenders. The methodology describes how, starting from an existing dataset, this has been enhanced to include the most recent knowledge from the MITRE ATT&CK database. It is then used to train a machine learning model that can classify unstructured cyber threat intelligence (CTI) sentences into mappings over the MITRE ATT&CK framework, allowing for more accurate and up-to-date analysis of attack strategies. Additionally, the methodology outlines how the model training and mapping processes work, explaining the workflow that enables SYNAPSE to classify and map attack patterns efficiently. Case studies were also conducted, including AI vs AI scenarios where SYNAPSE interacted with AI-driven attack tools. Additionally, the methodology presents DENDRITE, a traditional honeypot system, explaining its design and functionality. It details how DENDRITE has been developed to be equivalent to SYNAPSE in structure and configuration, allowing for direct comparative analysis.

Chapter 4 presented the experiments to evaluate SYNAPSE's performance. Three distinct experiments have been discussed, each designed to assess different capabilities. The first experiment focuses on real-world attackers interacting with SYNAPSE. It analyzes data such as IP addresses and reputations, geolocations, the number of connections, and connection duration. These findings provide a global perspective on attack origins, illustrating effectiveness in capturing critical intelligence. The second experiment involved controlled user interactions, comparing SYNAPSE with its static counterpart, DENDRITE. It measured how effectively the honeypot integrated with AI deceives attackers by simulating a natural environment. The results of this comparative analysis highlighted SYNAPSE's superior ability to mimic realistic system behaviour, making it more difficult for attackers to distinguish between the honeypot and a legitimate system. The third experiment examined AI-driven attacks. This scenario tested SYNAPSE's adaptability and ability to map complex attack techniques to the MITRE ATT&CK framework. The results demonstrated that the SYNAPSE-to-MITRE extension effectively and accurately classified attack techniques, providing defenders with actionable insights. Together, these experiments underscore the advantages of integrating AI into honeypots, showcasing SYNAPSE's ability to dynamically respond to and analyze a wide range of cyber threats. The results confirm that SYNAPSE enhances cybersecurity by transforming passive monitoring systems into proactive defence mechanisms.

5.1 Broader Implications of AI in Cybersecurity

In addition to the immediate benefits demonstrated by SYNAPSE, integrating AI into cybersecurity holds vast potential across various domains. This research advances honeypot technology and establishes a strong precedent for future developments in AI-driven defence mechanisms. AI's ability to learn and adapt in real-time offers applications in areas such as intrusion detection systems (IDS), automated vulnerability scanning, and threat hunting, where traditional security tools often struggle to keep up with rapidly evolving threats. Through the experiments conducted in this thesis, it has been demonstrated that incorporating AI into cybersecurity defences brings distinct advantages. AI can autonomously process vast amounts of data, recognize patterns, and adjust strategies faster than manual or static methods.

Furthermore, automatically mapping logs and unstructured cyber threat intelligence (CTI) to structured frameworks such as MITRE ATT&CK significantly enhances operational efficiency. The ability of SYNAPSE to classify attacker tactics and techniques in real-time, using machine learning models trained on the latest MITRE ATT&CK datasets, illustrates how AI can reduce the time required for threat analysis and response. Traditionally, mapping attacker behaviour to frameworks like MITRE ATT&CK is a manual, time-consuming process, requiring skilled analysts to examine logs and CTI. By automating this process, SYNAPSE saves time, enabling defenders to focus on response strategies rather than being engaged by data processing. This shift to automation can accelerate the speed at which organizations can detect, understand, and mitigate attacks, reducing the overall impact of a breach. The implications extend beyond SYNAPSE, offering a vision for AI-integrated security architectures that can evolve with the threats they face. As AI becomes more advanced, it could be applied to various cybersecurity functions, such as automated incident response, where AI-driven systems take predefined actions based on specific attack patterns. Additionally, AI could enhance behavioural analysis and predictive threat modelling, providing security teams with real-time alerts based on detecting anomalies that indicate potential threats. The successful demonstration of these concepts through SYNAPSE paves the way for future innovations in integrating AI with established cybersecurity frameworks like MITRE ATT&CK.

5.2 Challenges or Limitations

While SYNAPSE demonstrated significant advancements in integrating AI with honeypot technology, some notable challenges and limitations must be considered when deploying AI in critical cybersecurity applications. One primary concern is the risk of AI “hallucinations”, where generative models produce incorrect or misleading outputs that do not correspond to the input data. Critical applications requiring precise and reliable outputs can lead to errors in analysis, potentially causing misclassifications or overlooking key attack details. The consequences of such mistakes in cybersecurity, where accuracy is paramount, could result in misinformed defence strategies, delayed responses, or even undetected threats. Another limitation is the context window of AI models. Large language models (LLMs) are typically constrained by the context they can retain, which becomes problematic in long-running sessions where the prompt grows excessively large. Over time, as attackers interact with SYNAPSE over extended periods, the accumulated interactions may cause the AI to lose track of earlier context, leading to response inaccuracies. This loss can severely impact the continuity and consistency of AI-generated responses, limiting the effectiveness of the honeypot in understanding and adapting to evolving attacker strategies across multiple sessions. Moreover, the response time required for AI to compute answers poses a potential risk in time-sensitive applications. While AI can process and generate responses quickly, there is still an inherent delay, especially when the task involves complex computations. Systems that depend on instantaneous reactions, such as real-time attack mitigation or incident response, may find AI’s processing speed inadequate, creating a bottleneck that could allow threats to escalate.

A further challenge lies in training AI models to achieve high-performance metrics. Training a machine learning model, which maps unstructured data to structured frameworks like MITRE ATT&CK, requires a well-curated and large dataset. In the context of SYNAPSE, the AI must generate sentences summarising logs to align with the patterns in the data used for training. Any inconsistency between the generated data and the training set can negatively impact the model’s ability to map accurately. Obtaining and preparing such high-quality datasets is a resource-intensive task that demands significant expertise and time. Furthermore, ensuring the generated sentences are semantically and structurally similar to the training data is challenging, especially in real-world scenarios where attack patterns vary widely. Given the variability in unstructured cyber threat intelligence (CTI) data structure, ensuring that the AI consistently maps these logs to the correct tactics and techniques requires advanced training and rigorous testing. Achieving high precision and recall in these mappings is crucial for providing actionable insights but remains challenging due to the diversity of CTI formats and evolving attacker strategies.

Finally, managing ethical and legal considerations is another challenge when deploying AI in cybersecurity. As AI systems interact with attackers and collect data, there are concerns about data privacy, consent, and handling sensitive information. Additionally, deploying AI-driven honeypots raises questions about the ethical use of deception in cybersecurity defence. Navigating these moral and legal landscapes requires a clear understanding of regulations regarding how AI can be used and how data should be handled. Striking a balance between collecting valuable intelligence and maintaining ethical and legal compliance is an ongoing challenge in AI-based cybersecurity systems.

5.3 Applications of SYNAPSE in Cybersecurity

The findings of this thesis demonstrate how SYNAPSE can be applied in real-world cybersecurity environments, particularly in industries that are frequent targets of advanced persistent threats (APTs) or sophisticated cyberattacks. Sectors such as finance, healthcare, energy, and government agencies, where attackers often employ advanced and evolving tactics, would benefit from SYNAPSE's adaptive capabilities. Organizations could deploy SYNAPSE as part of their threat intelligence and monitoring strategies, using its generative AI-driven functionality to engage attackers, gather valuable data, and continuously improve their defences. SYNAPSE's integration with the MITRE ATT&CK framework offers an efficient approach for real-time mapping of adversary tactics, techniques, and procedures (TTPs). This mapping enables security teams to recognize attack patterns as they unfold, providing immediate insights into potential attack vectors. By automating the mapping process, SYNAPSE significantly reduces the time required for human analysts to interpret attack data, thereby enhancing the speed and efficiency of response strategies. This feature is precious in large-scale enterprise environments, where manual threat analysis may delay critical responses to complex cyberattacks. Additionally, SYNAPSE's ability to simulate realistic environments makes it an invaluable tool for deception-based security strategies. Unlike traditional honeypots, SYNAPSE can dynamically evolve based on attacker behaviour, making it more difficult for adversaries to identify and evade the honeypot. In real-world settings, this capability can delay and divert attackers from actual production systems, buying valuable time for security teams to respond.

SYNAPSE's flexibility as a cloud-based or on-premises system makes it scalable to a wide range of organizations. It can be deployed in cloud environments for greater scalability, allowing multiple instances to monitor distributed networks or cloud-based applications. Alternatively, it can be implemented in on-premises infrastructures where strict compliance, privacy, or legal requirements must be adhered to, such as in regulated industries like banking or healthcare. Another potential application of SYNAPSE lies in its ability to be integrated into security operations centres (SOCs). Within a SOC, SYNAPSE could function as an active component of a cyber threat intelligence platform, autonomously engaging with and analyzing attacks and delivering high-fidelity intelligence that can be acted upon quickly. The data SYNAPSE collects could also be invaluable for forensic analysis post-breach, providing detailed logs and timelines that help investigators understand the full scope and methodology of the attack.

Beyond organizational security, SYNAPSE could also be applied in educational settings as a training tool. Its AI-driven adaptability allows for creating simulated cyberattack environments where cybersecurity professionals can practice and refine their incident response strategies. By exposing trainees to various attack vectors, including AI-driven adversaries, SYNAPSE can help build more robust security teams capable of responding to complex, evolving threats in real-time. In summary, SYNAPSE represents a next-generation security tool that can be effectively applied in various real-world scenarios. Its adaptive AI capabilities and automated threat intelligence mapping make it a powerful asset in protecting against current and future cyber threats, enhancing organizational resilience and reducing response times in critical incidents.

5.4 Advancing SYNAPSE: Future Improvements

Regarding future improvements, SYNAPSE can evolve in several critical areas to enhance its efficiency, adaptability, and realism. One potential avenue is leveraging more advanced AI models, such as the forthcoming "GPT-5", or specialized AI models that can be run locally. Running models locally could theoretically reduce latency by minimizing data transmission times and offering faster, more immediate responses. However, this benefit must be weighed against the significantly lower computational power available on local systems compared to cloud-based infrastructure. Local environments, especially for smaller organizations, typically lack the hardware necessary to run highly complex AI models efficiently, leading to limitations in the size and complexity of the models that can be deployed. For example, a locally deployed model might reduce latency. Still, its overall performance could suffer if the system cannot handle the computational load, resulting in less accurate responses or slower processing times. Moreover, the energy consumption and hardware costs associated with running advanced AI models locally would likely be substantial, making this approach more suitable for large enterprises with robust infrastructures.

Another essential upgrade would involve addressing AI "hallucinations", where the model generates incorrect outputs. Future iterations of SYNAPSE could incorporate better model fine-tuning or error-detection algorithms to minimize these inaccuracies. Additionally, increasing the model's context window would improve SYNAPSE's capacity to handle long sessions with attackers without losing track of prior interactions. This is especially important in advanced cyberattacks that span extended periods, requiring the AI to maintain context over multiple stages of the engagement. In terms of functionality, expanding the number of simulated services beyond SSH and MySQL would make SYNAPSE more attractive to a broader range of attackers. Adding popular services like FTP, HTTP, or SMTP would increase the honeypot's appeal and provide richer data for analysis. Feedback from experiment two (see Section 4.2) also indicated that SYNAPSE's terminal simulation could benefit from usability improvements, such as introducing a coloured terminal interface and tab completion. These are standard features in real-world terminal environments and would significantly enhance SYNAPSE's realism, making it even harder for attackers to distinguish it from legitimate systems.

Another critical area for future development is improving mapping accuracy to the MITRE ATT&CK framework. The current setup generates multiple mappings to increase coverage, but future versions of SYNAPSE should focus on developing the precise number of mappings based on the attack's characteristics. By tailoring the mapping process, SYNAPSE could better account for attacks correlating to multiple techniques within MITRE ATT&CK, ensuring that only relevant tactics are highlighted. Furthermore, adopting a more advanced machine learning model could help enhance classification accuracy, allowing for better alignment between unstructured logs and MITRE techniques. To maintain the relevance of its mappings, SYNAPSE would also benefit from automating and updating its dataset with the latest information from the MITRE ATT&CK framework. As new tactics, techniques, and procedures (TTPs) are identified and added to the database, SYNAPSE's model should be continuously updated to reflect these changes. Automating this update process ensures that SYNAPSE remains current with the latest attack strategies, allowing for accurate and up-to-date threat intelligence analysis.

In conclusion, SYNAPSE's future evolution lies in its ability to integrate more advanced AI models, improve operational speed and accuracy, simulate a more comprehensive set of services, and refine its threat mapping capabilities. Enhancing the system's realism, responsiveness, and adaptability will allow SYNAPSE to continue being a cutting-edge cybersecurity defence tool while ensuring it remains effective in the face of evolving cyber threats. With ongoing improvements, SYNAPSE could outperform traditional honeypots and redefine how organizations approach cybersecurity, making it a proactive and intelligent component of any defence strategy.

Appendix A

Tab Auto-Completion

In this appendix chapter, we explore the implementation of **tab auto-completion** in a simulated Linux terminal environment, part of a honeypot system integrated with generative AI. The purpose of this feature is to mimic the behaviour of a typical Linux terminal, allowing users to type partial commands and have the system suggest or complete the input automatically. The concept of tab auto-completion is widely adopted in many command-line interfaces (CLIs), particularly in Unix-like operating systems, where it aids users by suggesting possible commands, file names, or options based on the initial input. This improves efficiency and reduces the likelihood of user error. In the context of a honeypot, tab auto-completion can play a dual role. It enhances the realism of the simulated environment and allows the system to analyse user behaviour in real-time, using the generative AI component to provide dynamic responses. The honeypot can gather valuable data for analysis, such as command usage patterns and potential malicious activities, by tracking the user's interaction with the auto-completion feature.

A.1 readline Module

The "readline" module is a core Python library that provides line-editing and history capabilities to interactive command-line interfaces. It enables advanced input handling, allowing users to navigate through input history, edit lines, and use shortcuts. A crucial aspect of readline is its ability to provide tab auto-completion, which offers suggestions for commands, filenames, or parameters, creating a seamless and user-friendly experience. The primary mechanism of tab auto-completion in readline involves binding the "tab" key to trigger a "completer" function, which generates possible suggestions based on the current input. By leveraging the readline module, we can simulate the experience of an actual Linux terminal, allowing users to dynamically type partially completed commands and receive intelligent suggestions.

A.1.1 input()

In Python, the "input" function captures user input from the command line. By default, it waits for user input and returns the typed string once the user presses the enter key. However, without additional features, *input()* lacks the sophisticated line-editing and auto-completion functionalities that users expect in a terminal environment. This is where the readline module enhances its functionality. We enable extended features by integrating readline with input(), making the terminal more interactive and user-friendly. This is especially important in a honeypot terminal simulation, where we aim to recreate a realistic environment to encourage typical user behaviour, which the system can monitor and analyse.

A.1.2 set_completer() and completer

A vital feature of the readline module is its ability to allow for custom tab auto-completion through the *set_completer()* function. It accepts a Python function as its argument, known as the "completer", that will determine the suggestions provided to the user when they press the tab key. It takes two parameters:

- **Current Text**: the partial input the user is trying to complete.
- **State**: an integer used to cycle through possible completions.

The completer function returns a list of possible completions based on the user input and state. This is how *set_completer()* and *completer()* are bound:

```
readline.set_completer(completer)
```

A.1.3 parse_and_bind()

The *parse_and_bind()* function in the readline module is used to bind specific keys to commands or functions. In our case, it binds the tab key to the completion function, allowing users to trigger auto-completion by pressing the tab. This function takes a string argument in the format *key: action*, where "key" is the key to be bound and "action" is the function or command to be executed. For instance:

```
readline.parse_and_bind('tab: complete')
```

binds the tab key to the completion function. As a result, when the user presses the tab key while typing a command in the terminal, the *set_completer()* function is triggered, and the terminal suggests possible completions. Binding keys like this is crucial for creating an intuitive terminal simulation. It ensures that the simulated environment behaves as a user would expect in an actual Linux terminal, thereby enhancing the realism of the honeypot.

A.2 Integration with Generative AI

Implementing tab auto-completion in a simulated honeypot environment introduces several unique challenges. The goal is to replicate the behaviour of a Linux terminal as closely as possible while integrating generative AI to provide intelligent, context-aware suggestions. One key challenge is creating an auto-completion system that balances realism with functionality. In a natural environment, the system suggests commands, file names, or arguments based on the current context. In the honeypot, we can enhance this by using generative AI to offer dynamic suggestions based on user behaviour, the current session, or even anticipated malicious activity. For example, the AI might recognise when a user attempts to execute a series of related commands and offer completions that guide them toward specific actions. This creates a more engaging and believable experience for the user while also providing the honeypot system with valuable data about the user's behaviour.

A.2.1 SYNAPSE implementation

In the case of SYNAPSE, the implementation of tab auto-completion has been achieved with partial success. SYNAPSE utilises a custom completer function that integrates generative AI to simulate the auto-completion process. The system begins with a prompt feeding the AI, such as

Emulate the tab auto-completion of a Linux terminal . . .

Which follows the principles of "Persona Prompting". This method allows the AI to generate suggestions when the user types a partial command or word. However, the current implementation only works in a limited capacity. Specifically, the AI generates a single completion based on the starting fragment of the word provided by the user. While this does enable essential word completion, it lacks a deeper contextual awareness. The AI cannot access or know the simulated environment's dynamically generated directories, files, or content structures. Without this context-linking mechanism, SYNAPSE's AI provides only generic word completions rather than precise suggestions related to the user's immediate environment. For example, it cannot complete directories or file names based on the simulated environment's state, which is a significant limitation compared to a fully realised tab auto-completion system.

To enhance the functionality and make the AI completions more accurate and context-aware, the system would need to be expanded to include details about the dynamically generated directories and files in the prompt. By integrating this information into the AI's prompt, the system could provide relevant completions that reflect the current state of the simulated terminal. This would create a more robust and realistic simulation, offering a more immersive user experience and increasing the utility of the auto-completion feature. In summary, while SYNAPSE has successfully implemented a basic version of tab auto-completion using generative AI, further integration of contextual data from the simulated environment would be necessary to achieve complete, accurate, and context-sensitive completions. This is a potential avenue for future development to enhance the realism and effectiveness of the simulated terminal experience.

Appendix B

System Logs Analysis

System logs are critical in monitoring, diagnosing, and securing an operating system. In a typical Ubuntu environment, logs such as *auth.log*, *syslog*, and *kern.log* provide valuable information about authentication events, system messages, and kernel activity. However, the manual analysis of these logs can be time-consuming and complex, requiring deep expertise to extract meaningful insights that can aid in security monitoring and system administration.

This case study explores the use of AI to automate the analysis of these critical system logs. The goal of passing the main logs of an Ubuntu system to a generative AI model is to enable the AI to extract valuable information that would assist a professional in identifying patterns, potential security risks, and critical system events. This approach aims to streamline the log analysis process, making it more efficient and accessible for IT administrators and cybersecurity professionals. Applications of this analysis include the detection of common security threats, such as brute force attacks, denial-of-service (DoS) attacks, and unauthorized access attempts. Additionally, by parsing through large volumes of system data, AI-based analysis can identify performance bottlenecks, system misconfigurations, or hardware failures. By automating these tasks, AI offers a powerful tool for reducing the manual effort required to maintain system security and operational health, providing timely insights that can enhance system performance and security defences.

B.1 Analysis Workflow

The system logs analysis processes each log file individually, following a structured approach to extract meaningful information. First, the log file is fetched from the */var/log* folder, focusing on logs like *auth.log*, *syslog*, and *kern.log*. To manage large log files efficiently and avoid exceeding the AI's context processing limits, the most recent N lines of the log are selected for analysis, where N is a configurable parameter that can be adjusted before starting the process. Once the appropriate log lines are chosen, they are passed to the AI for analysis. Thanks to prompt engineering, the latter identifies and extracts vital patterns, events, and essential details from the log entries. For example, in the case of *auth.log*, the AI might group login attempts by IP address, flagging any suspicious behaviour or security threats. The AI produces a schematic output, which focuses on relevant information without any unnecessary explanations or formatting.

After completing the analysis, the AI-generated output is saved in a *.txt* file. This file contains the structured analysis, making it easy for analysts to review and understand critical aspects of the system logs. Each log is processed this way, ensuring that the relevant data from each file is extracted, stored, and ready for further review by system administrators or security experts. This process not only automates the traditionally manual task of log analysis but also ensures that the most important details are highlighted efficiently.

Appendix C

User's Manual

This manual is a comprehensive guide for end users, detailing system requirements, installation instructions, and a usage guide for SYNAPSE. It aims to help users effectively set up and operate the honeypot system, offering troubleshooting guidance and additional information to explore SYNAPSE's capabilities further.

System Requirements

Operating System:

- Linux OS: Ubuntu[1] (tested on version 22.04 LTS).

Software Dependencies:

- Python[2]: version 3.8 or higher.
- Git[3]: required for cloning the project repository.
- Pip[4]: required for installing Python dependencies (version 20.0 or higher).

API Keys:

- OpenAI API Key[5]: required for integrating the "gpt-4o" model.
- VirusTotal API Key[6]: required for checking IP and domain reputations.

Cloud Platform (Optional):

- SYNAPSE can be hosted on cloud platforms, such as Oracle OCI[7], to ensure scalability and ease of remote access. Follow the cloud provider's guidelines for setting up a virtual machine (VM) with Ubuntu.

[1] `https://www.ubuntu-it.org/`

[2] `https://www.python.org/`

[3] `https://git-scm.com/`

[4] `https://pypi.org/project/pip/`

[5] `https://openai.com/index/openai-api/`

[6] `https://www.virustotal.com/gui/my-apikey`

[7] `https://www.oracle.com/it/cloud/`

Installation Instructions

1. Clone the Repository

To download the SYNAPSE project, open a terminal and clone the GitHub repository:

```
git clone https://github.com/eneagizzarelli/SYNAPSE.git
```

2. Install Python Dependencies

Navigate to the project directory and install the required Python packages using `pip`:

```
cd SYNAPSE/
pip install -r requirements.txt
```

3. Setup API Keys

Create a `.env` file in the parent directory of the project and add your API keys for OpenAI and VirusTotal in the following format:

```
OPENAI_API_KEY='YOUR KEY'
VIRUSTOTAL_API_KEY='YOUR KEY'
```

Note 1: in the original configuration, the `SYNAPSE` project folder has been cloned under the specific path `/home/enea/SYNAPSE`. Every script/source file in this project refers to other scripts/source files using the above absolute path as a base path. If you plan to use an alternative configuration (such as a different location or user), remember to change the paths and replace `enea` everywhere in the code.

4. Run the Configuration Script

Copy the `configSYNAPSE.sh` script from the `scripts/` folder outside the `SYNAPSE` directory. After assigning the necessary permissions, run the script:

```
chmod +x configSYNAPSE.sh
./configSYNAPSE.sh
```

This will complete the configuration of SYNAPSE, creating the necessary folders, downloading the GeoLite2 database, and assigning ownership and permissions to the user `enea` (or the user you have specified).

5. Configure the SSH Server

Modify your `/etc/ssh/sshd_config` file to run the `startSYNAPSE.sh` script (after assigning the necessary permissions) and turn off certain SSH parameters not handled by the SYNAPSE code. These modifications should apply whenever the user `enea` (or the user you have specified) connects to your machine using SSH:

```
Match User enea
   ForceCommand /home/enea/SYNAPSE/scripts/startSYNAPSE.sh
   X11Forwarding no
   AllowTcpForwarding no
   AllowAgentForwarding no
   PermitTunnel no
   PermitOpen none
```

Note 2: if you are hosting the code on a VM like Oracle OCI and you want to allow password authentication, remember to modify your /etc/ssh/sshd_config.d/60-cloudimg-settings.conf file by setting:

```
PasswordAuthentication yes
```

After making these changes, restart the SSH service:

```
sudo systemctl restart sshd
```

Usage Guide

Adopting the configuration mentioned above will run SYNAPSE's simulated terminal instead of the real one whenever user enea (or the user you specifically decided) connects to your SSH server.

1. SYNAPSE-to-MITRE

To start the SYNAPSE-to-MITRE extension, assign the necessary permissions to the corresponding script in the scripts/ folder and run it:

```
chmod +x startSYNAPSE-to-MITRE.sh
./startSYNAPSE-to-MITRE.sh
```

If the AI detects an attack based on the session data, this script will convert the classification files into attack files containing the corresponding MITRE ATT&CK object content.

2. Dataset Build

To re-build the dataset used to train the SYNAPSE-to-MITRE model:

1. Replace the capec or enterprise-attack databases in the SYNAPSE-to-MITRE/data folder with your preferred versions.
2. Ensure the file and folder names remain unchanged.
3. Build the dataset by executing:

   ```
   chmod +x startDatasetBuild.sh
   ./startDatasetBuild.sh
   ```

3. Model Training

To re-train the model with the newly created dataset:

```
chmod +x startModelTraining.sh
./startModelTraining.sh
```

4. AI vs AI

To run the AI vs AI interactions, move to the use_cases folder and run the preferred script:

```
python3 AI_vs_SYNAPSE_basic_interaction.py
```

or

```
python3 AI_vs_SYNAPSE_attacker_interaction.py
```

The second script's number of attacks can be modified by changing variable attacks_num in the code.

5. System Log Analysis

To perform a system log analysis of Linux OS logs using AI:

```
chmod +x startSystemLogsAnalysis.sh
./startSystemLogsAnalysis.sh
```

This script analyzes `auth.log`, `kern.log`, and `syslog` files using generative AI and provides a report summarizing system events.

Troubleshooting

If you experience an error like

```
Resource X not found
```

and, further on,

```
>>> nltk.download('X')
```

when using SYNAPSE-to-MITRE, please try the following command:

```
python3 -m nltk.downloader X
```

This issue generally occurs for resources such as `punkt` and `wordnet`.

Related Resources

To consult the DENDRITE manual, please refer to the following link: https://github.com/eneagizzarelli/DENDRITE

Appendix D

Programmer's Manual

This manual is a detailed guide for developers working with SYNAPSE. Its goal is to help them understand the software's architecture, explain how the different modules interact, and offer guidance for extending or contributing to the project.

Software Architecture

SYNAPSE comprises different interconnected modules, each vital in its functionality. This section outlines the purpose of each module, breaking down the Python files that compose them and highlighting the interactions between different components. The goal is to give developers a solid foundation for navigating the codebase, enabling them to understand how the system operates quickly.

```
SYNAPSE
|__data/
|__logs/
|__prompts/
|__scripts/
|__src/
|__SYNAPSE-to-MITRE/
|__system_logs_analysis/
|__use_cases/
```

Note: The `configSYNAPSE.sh` script can be used to configure the SYNAPSE project entirely. As described in the User's Manual (see Appendix C), the installation process starts by cloning the repository, after which `configSYNAPSE.sh` must be run to set up the project properly. This script can delete any previous root folder, clone the project again, and create the necessary `logs/` and `data/` directories. Additionally, the code automatically calls the `downloadGeoLiteDB.sh` script that removes any existing version of the GeoLite2 database and replaces it with the latest one, storing it inside the `data/` folder. The database can be kept up-to-date by scheduling the `downloadGeoLiteDB.sh` script alone in a cron task. Finally, the `configSYNAPSE.sh` script assigns the user `enea` the necessary ownership of the project folder and adjusts permissions accordingly.

SYNAPSE Module

The **SYNAPSE** module forms the project's core, containing the code necessary to simulate a Linux OS terminal and MySQL service by leveraging AI. It integrates the generative AI component to provide intelligent responses, mimicking actual system behaviours. SYNAPSE handles all interactions and data flows, making it crucial to the system's overall functionality. The code for the SYNAPSE module is located inside the `src/` folder at the project's root.

```
src/
├── ai_requests.py
├── client_data.py
├── initializations.py
├── simulations.py
└── SYNAPSE.py
```

SYNAPSE.py

The `SYNAPSE.py` code is the main program launched by the `startSYNAPSE.sh` container script, which is triggered by the *ForceCommand* directive inside `sshd_config` when a user connects via SSH. The primary responsibilities of `SYNAPSE.py` include managing client information, interactions and session handling. It communicates with `client_data.py` to initialise client data, increment the number of connections, and append session duration. Next, `SYNAPSE.py` calls functions from `initializations.py` to load the terminal prompt configuration and, if not the first connection, retrieve previous session commands and responses. This step ensures that all necessary data is prepared before the AI-driven terminal simulation starts. Once initialised, the terminal simulation is executed by calling the `terminal_simulation()` function from `simulations.py`. Additionally, `SYNAPSE.py` handles connection termination by responding to user input such as *CTRL+C* or *CTRL+D*, displaying a *logout* message when appropriate.

client_data.py

`client_data.py` is responsible for managing extracted client data. Upon loading, it retrieves the client IP address as a global variable by calling the `get_client_ip()` function, which extracts the IP from the `SSH_CLIENT` environment variable. The `initialize_client_data()` function, invoked by `SYNAPSE.py`, creates a folder for the client IP inside the `logs/` directory and initialises a JSON data structure. This includes IP information extracted via the `get_client_ip_info()` function, port details from the `SSH_CLIENT` variable, and geolocation data by calling `get_client_geolocation()`. These details, along with the number of connections (initialised to 0) and session durations (initialised as an empty array), are stored in the client's file within its folder. This initialisation is skipped for subsequent connections, and the main program only increments the connection count and logs the session duration.

The `get_client_ip_info()` function retrieves information about the client IP by calling VirusTotal APIs. It extracts essential information, such as `total_votes`, `whois`, `reputation`, `last_analysis_stats`, `regional_internet_registry`, and `as_owner`, and stores them in a JSON object. Similarly, the `get_client_geolocation()` function uses the GeoLite2 City database to gather details such as `country`, `region`, `city`, `postal_code`, `location`, and `continent`, which are also stored in a JSON object.

The `increment_client_number_of_connections()` function increments the number of connections by one at the start of each session. The `write_client_session_duration_in_seconds()` function calculates session duration by subtracting the start time from the end time and appends it to the array of durations in the client's data file. Lastly, the `get_count_classification_history_files()` function is a utility that provides the current number of classification files and is called by `simulations.py`.

initializations.py

The `initializations.py` code manages initialisations regarding prompts, arguments, and messages for the AI in both the terminal and MySQL services offered by SYNAPSE. The code operates almost equally for both functionalities. The `load_terminal_prompt()` and `load_mysql_prompt()` functions load the `terminal_personality.yml` or `services_personality.yml` files from the `prompts/` directory, respectively, if it is the first connection.

```
prompts/
├── terminal_personality.yml
└── services_personality.yml
```

If it is not the first connection, these functions load the previous terminal or MySQL histories from the client's IP directory inside the `logs/` folder. These files contain the respective prompt and logs of all commands and responses of earlier sessions.

The `parse_terminal_argument()` and `parse_mysql_argument()` functions receive the prompts mentioned earlier. They add the terminal or MySQL personality argument to the parser, using the terminal prompt or MySQL prompt as the default value. For the terminal, the AI is instructed to use the current date and IP address for the last login message, while for MySQL, the AI adopts a random ID for log in. The `load_terminal_messages()` and `load_mysql_messages()` functions generate the `messages` structure that is passed to the AI, ensuring that it is correctly formatted with *role* and *content* fields for OpenAI API calls. These functions initialise the `messages` variable with the prompt obtained earlier.

simulations.py

The `simulations.py` code manages the terminal simulation and initiates and manages the MySQL one. The `terminal_simulation()` function oversees the entire terminal simulation. It runs an infinite loop simulating the exchange of commands and responses between the user and the terminal. Within this loop, the last command issued by the user is extracted from the messages and parsed to check if it is an *exit* command, a *clear* command, or a *mysql* command. If none of these conditions apply, the command is sent to the AI by calling the `generate_response()` function from `ai_requests.py`. The AI's response is appended to the messages and written to the main terminal log file in the respective directory inside `logs/`. Additionally, the commands and responses for the session are stored in a separate `classification_history` file located in the same client's IP folder. The `SYNAPSE-to-MITRE` module will later analyse this file. Later in the simulation, the user is prompted to input a command, which is logged in all the files above and updated in the messages variable. The cycle then restarts.

The simulation gracefully closes the connection if the user inputs an *exit* command. If a *clear* one is issued, the system's `clear` call is executed. In the case of a *mysql* command, the `run_mysql_simulation()` function is called. This function checks the required parameters for authenticating to the MySQL service, such as *-u*, *-p*, and *-e*. If any issues are found, appropriate error messages are displayed. Once the parameters are validated, the MySQL prompt is loaded using functions from `initializations.py`, and the messages variable for MySQL is set up. The MySQL simulation is then initiated using the `mysql_simulation()` function, which ensures proper session handling and graceful exits back to the terminal simulation when the user decides to leave the MySQL service.

The `mysql_simulation()` function operates similarly to the `terminal_simulation()`, including handling cases where the user wants to execute an inline MySQL command from the terminal with the *-e* flag. The MySQL loop terminates in this scenario after the AI processes the command. The remaining code in `simulations.py` manages AI responses, ensuring that outputs are corrected when necessary. For instance, delays between output lines are added when executing commands, such as `ping`, to simulate natural system behaviour.

ai_requests.py

The `ai_requests.py` code handles communication with the AI by leveraging OpenAI APIs. The `generate_response()` function generates an AI response based on the prompt passed to it. After receiving the AI's output, the function parses and cleans the response to ensure it is well-structured. It then builds the assistant's message, which is returned to the caller. Additionally, the `generate_tab_completions()` and `completer()` functions are used to implement tab autocompletion. These functions work with the AI and the `readline` module to provide intelligent autocompletion suggestions based on the context of the user's input.

SYNAPSE-to-MITRE Module

The **SYNAPSE-to-MITRE** module is an extension of the SYNAPSE project. It enables mapping classification files generated from SYNAPSE interactions to the MITRE ATT&CK framework using AI and ML technologies. This module can be activated as needed and provides functionality to map observed behaviours to known tactics, techniques, and procedures (TTPs) within the MITRE ATT&CK framework. In addition to the classification mapping, the module includes code for re-generating the dataset used for training and re-training the machine learning model.

```
SYNAPSE-to-MITRE/
├── data/
│   ├── capec/
│   ├── enterprise-attack/
│   └── dataset.csv
├── ml_model/
│   └── MLP_classifier.sav
├── build_dataset.py
├── classification.py
├── SYNAPSE-to-MITRE.py
└── training.py
```

SYNAPSE-to-MITRE.py

The `SYNAPSE-to-MITRE.py` script can be launched using the container script `startSYNAPSE-to-MITRE.sh`. It contains the core code for mapping SYNAPSE interaction logs to the MITRE ATT&CK framework. When executed, `SYNAPSE-to-MITRE.py` scans all the folders inside the `logs/` directory at the project root, processing one IP folder at a time. For each IP, the reputation is retrieved. Then, the classification history files stored for the various sessions of that IP are analyzed sequentially in chronological order. For each classification file, the reputations of IPs and domains that appear in the commands issued by the user are extracted using the `get_ips_and_domains_from_classification_file()`, `get_ip_reputation()`, and `get_domain_reputation()` functions from `classification.py`.

At this point, the `attack_happened()` function from `classification.py` is called to determine whether an attack has occurred. If no attack is detected, the classification file is removed. If an attack is confirmed, the mapping process continues. The `get_sentence()` function from `classification.py` is invoked to generate a summary sentence of the entire classification history file. Next, the `get_attack_objects()` and `print_attack_objects_to_file()` functions from `classification.py` are called to retrieve the three possible mappings related to that sentence and print them to three distinct files. These files are stored alongside the analyzed classification file. This procedure is repeated for each classification file of every IP that has interacted with SYNAPSE.

classification.py

The `classification.py` file contains all the functions responsible for the logic of the mapping process within the SYNAPSE-to-MITRE module. The `get_ips_and_domains_from_classification_file()` function processes the entire passed classification history file. Using regular expressions to detect IP addresses and domain names, the file is analyzed line by line to identify potential IPs and domains. The `get_ip_reputation()` and `get_domain_reputation()` functions then extract the reputation of the identified IPs and domains by leveraging the VirusTotal APIs.

The `attack_happened()` function receives the classification file and all the extracted reputations. Using the entire session history, this function prompts the AI to determine whether an attack occurred based on the commands, responses, and reputations. The AI's decision is generated by calling the `generate_response()` function from the `ai_requests.py` code in the SYNAPSE module. The `get_sentence()` function scans the entire classification history file and, once again using the `generate_response()` function, generates an unstructured Cyber Threat Intelligence (CTI) sentence summarizing the entire interaction. This sentence is designed to resemble the sentences in the dataset for training the model.

The `get_classifications()` function receives the generated sentence and classifies it into three possible ATT&CK ID labels. They are then passed to the `get_attack_objects()` function, which extracts the corresponding attack objects by leveraging the *mitreattack* Python library and the `enterprise-attack` folder inside the `SYNAPSE-to-MITRE/data/` directory. The `print_attack_objects_to_file()` function prints the content of these attack objects to files, storing them in an attack directory alongside the classification history file for future analysis. Additionally, the `rename_classification_history()` and `remove_classification_history()` functions are utility functions used to rename and remove classification history files, respectively.

build_dataset.py

The `build_dataset.py` script is used to recreate the entire dataset from scratch. The process can be initiated by running the `startDatasetBuild.sh` container script. To successfully re-build the dataset, the `capec/` and `enterprise-attack/` folders inside the `SYNAPSE-to-MITRE/data/` directory need to be updated with the latest versions. Once done, the script will re-generate the dataset and save it in the same directory under the name `dataset_new.csv`. The file must be renamed to `dataset.csv` to use the newly generated dataset when re-building the model, replacing the previous version.

training.py

Using a new dataset, the `training.py` script is used to re-train the machine learning model from scratch. The process can be initiated by running the `startModelTraining.sh` script. This program starts by loading the `data/dataset.csv` file and uses it to train an `MLPClassifier` model. Once the training is complete, the resulting model is saved in the `SYNAPSE-to-MITRE/ml_model/` directory. Before running the `training.py` script, deleting any existing model file from the `ml_model/` directory is essential to ensure that the new one replaces the old one.

Case Studies Modules

The following modules are project components primarily representing case studies conducted during different experiments. They showcase practical applications of the SYNAPSE system and demonstrate its behaviour in various scenarios. Among them is the **DENDRITE** project, an integral part of the case studies. Its code can be found at the following GitHub repository: https://github.com/eneagizzarelli/DENDRITE.

use_cases

The `use_cases` folder contains programs that simulate AI interactions with the SYNAPSE system. These programs, named `AI_vs_SYNAPSE_basic_interaction.py` and `AI_vs_SYNAPSE_attacker_interaction.py`, showcase different AI-driven interactions.

```
use_cases/
├── AI_vs_SYNAPSE_attacker_interaction.py
└── AI_vs_SYNAPSE_basic_interaction.py
```

The `AI_vs_SYNAPSE_basic_interaction.py` script leverages the *paramiko* library to connect to a locally running instance of SYNAPSE. It simulates the behaviour of a benign user, interacting with the honeypot by navigating directories, creating files, and performing similar tasks. The code operates in an infinite loop, which can be stopped by SYNAPSE or by the user by pressing

CTRL+C. In this code, the last command outputted by SYNAPSE is captured and passed to the AI using the `generate_response()` function, as previously described. The AI's response is cleaned and appended to the list of messages for the current interaction. The resulting message is then sent back to SYNAPSE, with a brief pause introduced to allow it to generate its response sequentially and ensure the interaction proceeds smoothly.

On the other hand, the `AI_vs_SYNAPSE_attacker_interaction.py` script simulates a more aggressive interaction. It operates under a different logic, starting with a *while* loop that continues until a pre-defined number of attacks, set in the `attacks_num` variable, is reached. Like the basic interaction, `paramiko` connects to SYNAPSE. An infinite loop is started to simulate terminal interaction within each attack iteration. It can be stopped by SYNAPSE, by the user (again using *CTRL+C*), or by the AI itself if it determines the attack has ended. If the AI concludes the attack, it prints "Finished", and the Python code breaks the loop. Additionally, the latter can be interrupted after a pre-defined number of AI-issued commands, currently set to 15, to prevent the AI from entering an infinite loop without terminating the interaction. The rest of the process mirrors the basic interaction, where messages are processed and sent to SYNAPSE, and responses are captured sequentially.

system_logs_analysis

The `system_logs_analysis` module allows for AI-powered analysis of system logs, specifically `auth.log`, `kern.log`, and `syslog`, providing a summary of the essential details and information extracted from them.

```
system_logs_analysis/
└── analyze_system_logs.py
```

The analysis process can be started by running the `startSystemLogsAnalysis.sh` script, which invokes the `analyze_system_logs.py` code. Within it, three variables control the total number of lines to read from each log file: `AUTH_NUM_LINES`, `KERN_NUM_LINES`, and `SYSLOG_NUM_LINES`. They can be modified to avoid overloading the AI with entire log files, which could be too large to process efficiently. The analysis follows the same sequence for each log: first `auth.log`, then `kern.log`, and finally `syslog`. While the process is the same, the prompt given to the AI is customized for each of them. This is necessary because each log type has different characteristics, and the AI needs to focus on different aspects depending on the log being analyzed. The respective file is copied from the `/var/log/` directory and is truncated to include only the last pre-defined number of lines. The cut file, along with a log-specific prompt, is then passed to the `generate_response()` function in `ai_requests.py`. The AI's analysis is subsequently written to an output file. This process is repeated for each log in sequence.

Adding Services

For developers interested in expanding the functionality of SYNAPSE, this section provides instructions on how to add new services. It offers clear steps to ensure extensions are smoothly incorporated into the existing structure without disrupting the system's stability.

The first step to add a new service is to modify the `services_personality.yml` file in the root folder's `prompts/` directory. This YAML file has the following structure:

```
services:
  mysql:
    prompt:
      ...
```

A new service can be added by specifying the name and defining a new `prompt` under it. The latter should describe how the AI should behave while simulating the new service. It is essential to use effective prompt engineering techniques, such as Persona prompting and Few-Shot prompting, to achieve optimal performance.

After defining the prompt for the new service, several functions in `initializations.py` must be modified or added to support the new service. Specifically, the following functions should be implemented to follow the same structure as those for `terminal` and `mysql`:

- `load_service_prompt()`: loads the prompt for the new service, similar to how terminal and MySQL prompts are loaded.
- `parse_service_argument()`: parses the service argument from the command input to provide the correct personality and behaviour to the AI.
- `load_service_messages()`: prepares the message structure to be passed to the AI, following the format for terminal and MySQL messages.

Once the necessary functions are implemented in `initializations.py`, the `simulations.py` file must be updated. In the `terminal_simulation()` function, glue code should be added to capture whether the user has input a command to switch to the newly added service. If so, a new `service_simulation()` function should be defined, following the same pattern as existing service simulations, like `mysql_simulation()`. This glue code should manage all cases the AI cannot handle, allowing for a seamless transition between terminal and service simulations. It should behave closely to a natural system, simulating all its essential functionalities. Additionally, the simulation should handle logging like the MySQL service, creating a dedicated log file for the new service's interactions. By following these steps, developers can ensure that new services are fully integrated into the SYNAPSE system, maintaining consistency in interactions, responses, and logging mechanisms.

Contributing to the Project

This manual encourages open-source collaboration by outlining how to contribute to the SYNAPSE project. It provides guidance for setting up the development environment and submitting contributions through pull requests, ensuring that all modifications follow the project's coding standards and maintain a structured development process.

To contribute to SYNAPSE, the first step is to fork the original repository on GitHub. This creates a personal copy of the project under the contributor's account. After forking, the repository can be cloned to the local machine for modification. Creating a new branch for each feature or bug fix is recommended, ensuring the main one remains clean. Once changes are made, they should be committed and pushed to the forked repository. Before submitting any changes, contributors must test the functionality of their modifications. This requires configuring both the OpenAI API and the VirusTotal API, which is essential for running and verifying core features within the SYNAPSE project. Once the changes have been tested, contributors should open a pull request from their forked repository to propose the changes to the original project. It should include a clear description of the modifications made and their purpose. Project maintainers may review the changes and request updates, which can be incorporated directly into the same pull request. Once approved, the pull request will be merged into the original project, combining the contribution into SYNAPSE. By following this process, developers can effectively collaborate on SYNAPSE and contribute to its growth while ensuring consistency and code quality.

Bibliography

[1] N. I. of Standards and T. (NIST), “Information security definition”, https://csrc.nist.gov/glossary/term/information_security

[2] T. U. of Tulsa, “Information security vs. cybersecurity: What’s the difference?”, https://online.utulsa.edu/blog/information-security-vs-cybersecurity/#:~:text=Information%20security%20protects%20a%20variety,systems%20and%20networks%20from%20cyberattacks.

[3] M. Nawrocki, M. Wahlisch, T. C. Schmidt, C. Keil, and J. Schonfelder, “A survey on honeypot software and data analysis”, 2016

[4] Fortinet, “What are honeypots (computing)?”, https://www.fortinet.com/resources/cyberglossary/what-is-honeypot

[5] G. O’Shea, “Cyberlaw 101: A primer on laws related to honeypot deployments”, 2007, GIAC Security Essentials Certification (GSEC) Gold Paper

[6] T. H. Project, “The honeynet project”, https://www.honeynet.org/about/

[7] N. Provos, “Honeyd”, https://www.honeyd.org/

[8] DinoTools, “Dionaea”, https://dionaea.readthedocs.io/en/latest/

[9] M. Oosterhof, “What is cowrie”, https://cowrie.readthedocs.io/en/latest/README.html#what-is-cowrie

[10] IBM, “What is ai?”, https://www.ibm.com/topics/artificial-intelligence

[11] S. Feuerriegel, J. Hartmann, C. Janiesch, and P. Zschech, “Generative ai”, Business & Information Systems Engineering, vol. 66, no. 1, 2024, pp. 111–126, DOI 10.1007/s12599-023-00834-7

[12] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, “A comprehensive overview of large language models”, 2024

[13] M. M. Maestre, I. Martinez-Murillo, T. J. Martin, B. Navarro-Colorado, A. Ferrandez, A. S. Cueto, and E. Lloret, “Beyond generative artificial intelligence: Roadmap for natural language generation”, 2024

[14] M. AI, “What is tokenization?”, https://docs.mistral.ai/guides/tokenization/

[15] DAIR.AI, “Prompt engineering guide”, https://www.promptingguide.ai/

[16] S. Feuerriegel, J. Hartmann, C. Janiesch, and P. Zschech, “Generative ai”, Business & Information Systems Engineering, vol. 66, September 2023, pp. 111–126, DOI 10.1007/s12599-023-00834-7

[17] C. Janiesch, P. Zschech, and K. Heinrich, “Machine learning and deep learning”, Electronic Markets, vol. 31, April 2021, pp. 685–695, DOI 10.1007/s12525-021-00475-2

[18] M. Corporation, “Mitre”, https://www.mitre.org/who-we-are/our-story

[19] M. Corporation, “Mitre att&ck”, https://attack.mitre.org/

[20] CISA, “Best practices for mitre att&ck mapping”, https://www.cisa.gov/sites/default/files/2023-01/Best%20Practices%20for%20MITRE%20ATTCK%20Mapping.pdf

[21] V. Orbinato, M. Barbaraci, R. Natella, and D. Cotroneo, “Automatic mapping of unstructured Cyber Threat Intelligence: An experimental study”, Proceedings of the 33rd IEEE International Symposium on Software Reliability Engineering (ISSRE), 2022

[22] M. A. Ferrag, F. Alwahedi, A. Battah, B. Cherif, A. Mechri, and N. Tihanyi, “Generative ai and large language models for cyber security: All insights you need”, 2024

[23] M. J. Hossain Faruk, H. Shahriar, M. Valero, F. L. Barsha, S. Sobhan, M. A. Khan, M. Whitman, A. Cuzzocrea, D. Lo, A. Rahman, and F. Wu, “Malware detection and prevention using artificial intelligence techniques”, 2021 IEEE International Conference on Big Data (Big

Data), December 2021, DOI 10.1109/bigdata52589.2021.9671434

[24] Google, "New gmail protections for a safer, less spammy inbox", https://blog.google/products/gmail/gmail-security-authentication-spam-protection/#:~:text=Gmail's%20AI%2Dpowered%20defenses%20stop,complex%20and%20pressing%20than%20ever.

[25] M. Sladic, V. Valeros, C. Catania, and S. Garcia, "SheLLM", July 2023, Software

[26] M. Sladic, V. Valeros, C. Catania, and S. Garcia, "Llm in the shell: Generative honeypots", 2024

[27] S.-H. Lee, A. Abdullah, N. Z. Jhanjhi, and S.-P. Kok, "Classification of botnet attacks in iot smart factory using honeypot combined with machine learning", PeerJ Computer Science, vol. 7, 2021, p. e350, DOI 10.7717/peerj-cs.350

[28] D. Fraunholz, M. Zimmermann, and H. D. Schotten, "An adaptive honeypot configuration, deployment and maintenance strategy", 2021