Analysis of the Capability and Training of Chat Bots in the Generation of Rules for Firewall or Intrusion Detection System

Bernardo Amaral Louro

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

(2º ciclo de estudos)

Orientador: Prof. Doutor Pedro Ricardo Morais Inácio

Co-orientador: Prof. Doutor João Bernardo Ferreira Sequeiros

Janeiro, 2025

Declaração de Integridade

Eu, Bernardo Amaral Louro, que abaixo assino, estudante com o número de inscrição M12317 do Mestrado de Engenharia Informática da Faculdade das Engenharias, declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o Código de Integridades da Universidade da Beira Interior.

Mais concretamente afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer, que em particular atendi à exigida referenciação de frases, extratos, imagens e outras formas de trabalho intelectual, e assumindo assim na íntegra as responsabilidades da autoria.

Universidade da Beira Interior, Covilhã 11/01/2025

Assinado por: Bernardo Amaral Louro
Num. de Identificação: BI15836043
Data: 14-01-2025 21:49:24 +00:00

Acknowledgements

This dissertation would not have been possible without the support of several people to whom I am extremely grateful.

To Professor Pedro Inácio, for his guidance, continuous help and availability, always willing to offer suggestions and solutions that allowed this work to succeed.

To Professor Bernardo Sequeiros, for his guidance and willingness to share his knowledge and advice.

To my friends, who always accompanied me during this phase and always offered their concern and encouragement.

At last, to my family, who have always encouraged me to work hard and try to succeed, while giving me everything I needed to do so.

The work described in this dissertation was carried out at the Secure and Intelligent Networked Software Systems Laboratory {////} **sins**-lab and at the Instituto de Telecomunicações - Delegação da Covilhã, as part of the research group in Network Applications and Services {///} **nas**-cv, located at the Universidade da Beira Interior, Covilhã, Portugal.

Resumo alargado

A recente evolução dos Grandes Modelos de Linguagem (designados em inglês por *Large Language Model* (LLM)) trouxe consigo um grande interesse pela Inteligência Artificial (designada em inglês por *Artificial Intelligence* (AI)). A capacidade dos LLM de gerarem texto encontra aplicabilidade em vários cenários, também devido à sua fácil acessibilidade, e revelam potencial para complementar o conhecimento humano ou ajudar a colmatar a sua falta em várias áreas técnicas, nomeadamente na informática, onde são muito utilizados na assistência ao desenvolvimento de código. Estes modelos, capazes de manter uma conversa com substância, são também capazes de criar informação falsa ou fictícia, cujas consequências variam consoante o cenário onde são aplicados.

Os LLM foram treinados com base num conjunto muito extenso de dados, o que lhes permite compreender e responder a várias questões e tarefas. Isto inclui a capacidade de traduzir linguagem natural em linguagens de programação, *scripts* ou configurações para ferramentas de cibersegurança. Por exemplo, pode ser pedido a um *chat bot* que gere uma regra *iptables* para evitar um ciberataque específico, dando apenas o nome do ataque e confiando na capacidade do *chat bot* para extrapolar a informação necessária para produzir uma regra, uma vez que são encontrados exemplos em fóruns especializados na *Internet* e que, consequentemente, poderão ter sido utilizados para o treinar. A regra pode até parecer sintaticamente correta, o que torna muito mais difícil avaliar se é apropriada ou útil.

No entanto, os *chat bots* atualmente disponíveis não foram explicitamente treinados em cibersegurança, e muito menos em algo ainda mais específico nesta área, como a geração de *firewalls* e Sistemas de Deteção de Intrusões (designados em inglês por *Intrusion Detection Systems* (IDS)), e as suas respostas podem não ser exatamente ideais para os objetivos de segurança pretendidos, uma vez que são um desafio mesmo para administradores de sistemas experientes, pois são específicas de uma área técnica que está sempre a mudar. O risco de gerar regras subóptimas ou erradas levanta preocupações sobre a fiabilidade da utilização de *chat bots* em contextos práticos de segurança. Inicialmente motivada por estas dúvidas, neste projeto foi realizada uma análise humana para explorar a capacidade dos *chat bots* atuais de escrever regras de *firewall* e IDS e, em seguida, contemplar a possibilidade de dar *fine-tune* a esses mesmos *chat bots* para esta tarefa específica. Os *chat bots* analisados neste trabalho foram o *Microsoft 365 Copilot*, o *ChatGPT 3.5*, o *Gemini 1.5 Pro*, o *LLaMA 2 7B*, o *Mistral 7B*, o *GPT4All Falcon*, o *Nous-Hermes* e o *Wizard*.

O problema abordado neste trabalho é a possível falta de capacidade dos *chat bots* atual-

mente disponíveis para traduzir pedidos em linguagem natural para regras de *firewall* ou IDS. Na sequência das notícias contínuas sobre informações imprecisas ou inventadas, há interesse em avaliar a capacidade dos *chat bots* atuais para traduzir pedidos em linguagem natural para regras de *firewall* e de IDS e analisar a sua capacidade de resposta a pedidos que seriam possivelmente feitos por pessoas com diferentes formações e níveis de conhecimento. Em seguida, esperando que os resultados não sejam assim tão bons, tentar-se-á dar *fine-tune* a um *chat bot* para se especializar na escrita de regras de *firewall* e IDS e, com isso, criar uma ferramenta que possa ajudar a proteger muitas pessoas (particulares ou, e.g., administradores de sistemas) que possivelmente não teriam os conhecimentos necessários para se protegerem a si próprias e também avaliar se o *fine-tuning* é uma forma viável de criar um *chat bot* com um objetivo realmente específico. Os principais objetivos desta dissertação serão, portanto:

1. A análise da capacidade dos *chat bots* na geração de regras para *firewall* e IDS a partir de linguagem natural;
2. O *fine-tuning* de um *chat bot*;
3. A comparação da capacidade entre os *chat bots* antes e após *fine-tuning*;
4. A criação de um conjunto de dados que será utilizado para efeitos de teste e *fine-tuning*;
5. O *fine-tuning* de um *chat bot* com diferentes abordagens e a criação de um ou vários modelos que possam ser usados como suporte para gerar regras de *firewall* e IDS.

A abordagem adotada para atingir os objetivos definidos anteriormente começou pela recolha de diferentes ataques que podem ser evitados por *firewall* ou IDS. Após esta recolha, foram preparados alguns *prompts* (nesta dissertação e dado o seu contexto, um *prompt* é uma instrução, escrita tipicamente em linguagem humana, para um *chat bot* com um LLM) para apresentar aos *chat bots*, tendo em consideração que foram feitos múltiplos *prompts* para o mesmo ataque, como se pessoas de diferentes níveis de conhecimento os escrevessem. Foi analisada, de seguida, a capacidade dos diferentes *chat bots*, não olhando apenas para a taxa de sucesso, mas também analisando mais detalhadamente os resultados dos diferentes tipos de *prompts* e *softwares* quando comparados uns com os outros. Para além disto, foram analisadas as diferentes abordagens de *fine-tuning* de acordo com a taxa de sucesso e foi feita uma análise mais profunda para verificar se as respostas corretas e erradas seguiam algum tipo de padrão. Por último, foram analisados e comparados os diferentes *chat bots* antes e após os processos de *fine-tuning* de acordo com a taxa de sucesso e outras métricas.

As principais contribuições desta dissertação serão:

- Um estudo sistemático da capacidade dos chat bots atualmente disponíveis para gerar regras de *firewall* e IDS;
- Aumentar a sensibilização para a utilização de *chat bots* para a geração automática de definições de segurança e para o facto de estes poderem fornecer informações erradas, devendo ser utilizados com o máximo cuidado e sempre sob supervisão;
- Um *dataset* de um conjunto de *prompts* que descrevem ataques e as regras correspondentes para *iptables* e *Snort*;
- Uma análise de diferentes abordagens ao *fine-tuning* com comparações e conclusões;
- Uma análise da diferença entre modelos antes e após *fine-tuning* com comparações e conclusões;
- Conclusão sobre se os modelos podem ser *fine-tuned* e obter resultados agradáveis;
- Um ou vários modelos que possam ser utilizados para ajudar os profissionais e os utilizadores de sistemas informáticos com diferentes formações e conhecimentos técnicos;
- A maioria das contribuições anteriores foi publicada num artigo científico com o objetivo de partilhar esses resultados com a comunidade.

O capítulo do estado da arte divide-se em conceitos e terminologias preliminares e em trabalhos relacionados. O capítulo do estado da arte começa com a introdução de conceitos com os quais é pertinente estar familiarizado e todas as tecnologias importantes que foram usadas ao longo da dissertação. Esta secção ajuda a dar uma compreensão preliminar a tudo o que será discutido nos capítulos que o seguem. No que diz respeito aos trabalhos relacionados, é mencionado que embora não tenham sido encontradas abordagens equivalentes específicas na literatura, o *SecBot* mostra uma inspiração relacionável, embora seguindo uma abordagem diferente e com objectivos finais diferentes (mas relacionáveis). O *CyberBench* e o *CyberInstruct* são duas ferramentas inovadoras, e que foram concebidas para melhorar a utilização de LLM no domínio da cibersegurança, embora não abordem as regras de *firewall* e de IDS como este trabalho o faz. Também foram analisadas técnicas de *fine-tuning* utilizadas para personalizar *chat bots* em diferentes domínios, mostrando a validade desta abordagem para melhorar a sua aplicabilidade em áreas específicas. Por último, foi feita uma pesquisa mais alargada para compreender como os *chat bots* estão a ser utilizados em diferentes contextos

de segurança. Esse capítulo identifica e explora vários trabalhos que reflectem a investigação nesta área, destacando as oportunidades que estas novas abordagens trazem e mencionam que *chat bots* e LLM podem ser ferramentas úteis para a cibersegurança, mas que a utilidade dos mesmos tem de ser cuidadosamente ponderada à luz dos potenciais problemas que podem surgir de uma utilização indevida ou de resultados incorretos.

O capítulo do método introduz o método utilizado durante a dissertação, dando uma melhor ideia do que teve de ser feito para a completar, uma melhor compreensão do que cada decisão tomada ao longo do estudo estava a tentar alcançar e uma melhor ideia do que fazer se alguém quiser replicar este método.

O capítulo dos resultados apresenta os resultados obtidos durante a dissertação. Este trabalho explorou quatro abordagens de *fine-tuning*, tendo cada uma delas alcançado algum grau de sucesso nos seus diferentes objectivos. A abordagem #1 teve uma taxa de sucesso de 89% e avaliou se o conhecimento obtido continuava a ser emitido quando a pergunta era organizada de forma diferente. A abordagem #2 teve uma taxa de sucesso de 61% e avaliou se o modelo conseguia ligar os conhecimentos entre dois pares diferentes de perguntas-respostas. A abordagem #3 teve uma taxa de sucesso de 79% e avaliou se o modelo conseguia gerar regras complexas a partir da aprendizagem de regras mais simples e genéricas. A abordagem #4 avaliou se o modelo conseguia identificar regras através de uma pergunta de escolha múltipla e obteve uma taxa de sucesso de 48% e 89%, consoante a ordem das escolhas. Estes resultados evidenciam o potencial de dar *fine-tune* a um modelo e o mesmo é apoiado pela métrica ROUGE, que também é provado neste capítulo.

O capítulo da conclusão e trabalho futuro enfatiza a incapacidade dos *chat bots* atuais de gerar regras e a melhora conseguida pelos modelos que foram *fine-tuned*. Para trabalho futuro é mencionado que mais testes têm de ser feitos, o número de ataques e sistemas tem de ser aumentado, terá de haver uma exploração maior de diferentes parâmetros de *fine-tuning*, poderão ser criados mais *datasets* para colmatar as regras que mais falham e expandir esta metodologia para outras áreas da cibersegurança.

Abstract

Chat bots have a lot of potential to complement human knowledge or help fill a lack of it in various technical areas, like cybersecurity, by providing a tool that could generate specialized rules for computer security systems, such as Intrusion Detection Systems (IDSs) and firewalls, from instructions in natural language. The preliminary evaluation conducted within the scope of this work proved that currently available chat bots are limited when it comes to generating correct and efficient rules and that extra attention is needed if their outputs are to be deployed in production systems, as the consequences can be severe. The chat bots evaluated were *Microsoft 365 Copilot*, *ChatGPT 3.5*, *Gemini 1.5 Pro*, *LLaMA 2 7B*, *Mistral 7B*, *GPT4All Falcon*, *Nous-Hermes* and *Wizard*. This document explores four fine-tuning approaches to address some of these limitations, with each of them achieving some degree of success on their different objectives. Approach #1 had a success rate of 89% and assessed if the knowledge obtained was still outputted when the question was arranged differently. Approach #2 had a success rate of 61% and assessed if the model could link knowledge between two different prompt-response pairs. Approach #3 had a success rate of 79% and assessed if the model could create complex rule from learning simpler and generic rules. Approach #4 assessed if the model could identify rules through a multiple choice question, in which it achieved 48% and 89% success rate depending on the order of the choices. From this work it can be concluded that fine-tuning is successful in improving the generation of firewall and IDS rules in chat bots and the results suggest that with some improvements and considerations, a specialized model can be achieved.

Keywords

Chat Bots, ChatGPT 3.5, Cibersecurity, Firewall, Gemini 1.5 Pro, GPT4All Falcon, Intrusion Detection System (IDS), Large Language Models (LLMs), LLaMA 2 7B, Microsoft 365 Copilot, Mistral 7B, Nous-Hermes, Wizard.

Contents

List of Tables and Figures

Acronyms

ACM Association for Computing Machinery

Adam Adaptive Moment Estimation

AI Artificial Intelligence

CCS Computing Classification System

CRC Cyclic Redundancy Checksum

DDoS Distributed Denial of Service

DMZ Demilitarized Zone

DNS Domain Name System

DoS Denial of Service

GPT Generative Pre-trained Transformer

GPU Graphics Processing Unit

HOIC High Orbit Ion Cannon

HTML HyperText Markup Language

ICMP Internet Control Message Protocol

IDS Intrusion Detection System

IP Internet Protocol

IPS Intrusion Prevention System

ISAKMP Internet Security Association and Key Management Protocol

IT Information Technology

LAND Local Area Network Denial

LCS Longest Common Sequence

LLM Large Language Model

LOIC Low Orbit Ion Cannon

LoRA Low-Rank Adaptation

NetBIOS Network Basic Input/Output System

NLP Natural Language Processing

PEFT Parameter-Efficient Fine-Tuning

PTQ Post-Training Quantization

QAT Quantization-Aware Training

RAM Random Access Memory

RAT Remote Access Trojan

ROUGE Recall-Oriented Understudy for Gisting Evaluation

R.U.D.Y R-U-Dead-Yet

SGD Stochastic Gradient Descent

SPARC Scalable Processor Architecture

SQL Structured Query Language

SSH Secure Shell

TCP Transmission Control Protocol

UDP User Datagram Protocol

VRAM Video Random Access Memory

WCCP Web Cache Communication Protocol

XSS Cross-Site Scripting

Chapter 1

Introduction

1.1 Scope and Motivation

This dissertation was produced in the context of the Master's program in Computer Science and Engineering at University of Beira Interior, as part of the requirements to obtain the degree. The underlying work described herein was mostly performed in the second semester of the 2023/24 academic year.

1.1.1 Motivation

The recent evolution in Large Language Models (LLMs) has brought with it a great interest in Artificial Intelligence (AI). The ability of LLMs to generate text finds applicability in various scenarios [1, 2, 3], due to their easy accessibility, and they show potential to complement human knowledge or help fill a lack of it in various technical areas, particularly in computer science, where it is used extensively to assist in code development [4]. These models, while capable of holding an engaging and fluid conversation, also have the possibility of creating false or fictional information [5, 6], the consequences of which will vary depending on where they are applied.

LLMs have been trained on a very extensive set of data, which enables them to understand and respond to various questions and tasks. This includes the ability to translate natural language into programming languages, scripts or cybersecurity tool configurations. For example, a chat bot could be asked to generate an *iptables* rule to prevent a specific cyberattack, giving only the name of the attack and relying on the ability of the chat bot to extrapolate the information needed to produce a rule, as examples are found in specialized forums on the Internet and would consequently have been used to train it. The rule may even seem syntactically correct, which makes it much harder to assess whether it is appropriate or useful.

However, currently available chat bots have not been explicitly trained on cybersecurity, let alone in something even more specific within the area such as generating firewalls and Intrusion Detection Systems (IDSs) rules, and their responses may not be exactly ideal for the intended security objectives, as they are challenging even for seasoned system administrators, being specific to a technical area that is always changing. The risk of generating sub-

optimal or erroneous rules raises concerns about the reliability of using chat bots in practical security settings. Initially motivated by these doubts, a human analysis was carried out in the underlying project that this dissertation is describing, to explore how well current chat bots are able to write firewall and IDSs rules and then proceeded by contemplating the possibility of fine-tuning these chat bots for this specific task.

1.1.2 Scope

The scope of this work falls in the intersection of the areas of Cybersecurity and Applied Artificial Intelligence, since it deals with the use and analysis of chat bots to help write rules for firewalls and IDSs. Under the 2012 version of the Association for Computing Machinery (ACM) Computing Classification System (CCS), a *de facto* standard for classification in computer science, the scope of this master's project, reflected in this dissertation, is defined by the categories listed below in order of importance:

- **Security and privacy - Systems security;**
- *Computing methodologies - Artificial intelligence - Natural language processing - Natural language generation.*
- Security and privacy - Network security - Firewalls;
- Security and privacy - Intrusion/anomaly detection and malware mitigation - Intrusion detection systems.

1.2 Addressed Problem and Objectives

This section presents the problem addressed in the first subsection, which discusses current limitations and expected issues that may arise from the use of LLMs for rule generation. The second subsection presents the main objectives that this work proposes to achieve.

1.2.1 Addressed Problem

The problem addressed in this work is the possible lack of ability of current chat bots to translate requests in natural language to firewall or IDSs rules. Following the continuous headlines made for inaccurate or made up information coming from free and readily available LLMs, there is an interest in assessing the ability of current chat bots in translating natural language requests into firewall and IDSs rules and analyze how well they respond

to prompts that would possibly be made by people with different backgrounds and different levels of knowledge. Following that, expecting subpar results, there will be an attempt to fine-tune a chat bot that will specialize on writing firewall and IDSs rules and, with that, create a tool that could possibly help people who work in Information Technology (IT) (e.g., system administrators) with little knowledge of cybersecurity to manage their systems, or even help ordinary people with simple tasks related to cybersecurity, such as configuring firewalls.

1.2.2 Objectives

Taking into account what was mentioned in 1.2.1 the main objectives of this dissertation are as follows:

- Analysis of the current capability of chat bots in the generation of rules for firewall and IDSs from natural language;
- Creation of a dataset that will be used for testing and fine-tuning purposes;
- Fine-tuning chat bots with different approaches;
- Analysis and comparison of capabilities between the pre-tuned and fine-tuned chat bots;
- Create one or multiple models that can be used as support to generating firewall and IDS rules.

1.3 Adopted Approach for Solving the Problem

The approach adopted to achieve the objectives defined above was as follows:

1. Collection of different attacks that can be prevented with firewalls or IDSs;
2. Preparation of some prompts to present the chat bots, taking into consideration multiple prompts for the same attack, as if people of different knowledge levels wrote them;
3. Analysis of the capabilities of the different chat bots, not by looking at success rate alone, but also by analyzing deeper how the different kinds of prompts and the different software fare when compared to each other;
4. Analysis of different fine-tuning approaches by success rate and deeper analysis to verify if the correct and wrong answers follow some kind of behaviour;

5. Analysis and comparison of the different pre-tuned chat bots and fine-tuned ones by success rate and metrics.

1.4 Contributions and Results

With this master's dissertation there were contributions such as showing how current chat bots handle the writing of different rules for firewall and IDSs and providing a fine-tuned chat bot that can, possibly, help people with different knowledge and backgrounds setup firewall and IDSs and show if fine-tuning chat bots for this kind of task is something worth investing in. Therefore, the contributed results from this master's dissertation are:

- A systematic study of the capacity of currently available chat bots to generate firewall and IDSs rules;
- Increase awareness on the use of chat bots for automated generation of security settings, and the fact that they can can provide incorrect information, and should be used with the utmost care and always under supervision;
- A dataset of prompts that describe attacks and the corresponding rules for the *iptables* firewall and the *Snort* IDS. This contribution is available in the form of a *Github* repository in `https://github.com/BernardoLouro/rgft`;
- An analysis of different approaches to fine-tuning with comparisons and conclusions;
- An analysis of the difference between pre-tuned and fine-tuned models with comparisons and conclusions;
- Conclusion on whether models can be fine-tuned and achieve adequate results;
- One, or multiple models that can possibly be used to help practitioners and computer system users with different backgrounds and technical knowledge;
- Most of those previous contributions were published in a scientific article [7] with the purpose of sharing these results with the community.

1.5 Organization of the Dissertation

This document is organized as follows:

- Chapter 1 - Introduction - introduces this manuscript and the project by presenting the scope, motivation, the addressed problem, the main objectives, the adopted approach that was taken to achieve them, the contributions and results, and the organization of the document;
- Chapter 2 - State of the Art - presents the State of the Art, which includes works and background, particularly *SecBot*, *CyberBench* and *CyberInstruct*, related works and previous studies about fine-tuning and other security settings in which chat bots are being used, and relevant concepts and technologies, particularly the preliminary concepts and terminology needed to understand the dissertation and important technologies used during this study;
- Chapter 3 - Method - presents the methods used for the data collection and for the fine-tuning and discusses the four different approaches taken for fine-tuning;
- Chapter 4 - Results - presents the results, particularly the results of the preliminary evaluation that was done, the results of each fine-tuning approach, comparisons between before and after fine-tuning and Rouge-1 scores between models;
- Chapter 5 - Conclusion and Future Work - presents the conclusions drawn from this dissertation, as well as the future work.

Chapter 2

State of the Art

2.1 Introduction

This chapter starts by introducing relevant concepts and important technologies in this work, specifically in sections 2.2 and 2.3. In section 2.2, concepts such as the definition of *firewalls*, IDSs, chat bots, fine-tuning and other related concepts, the rouge metric and some attacks are presented, as they are pertinent to be familiar with. Following that, in section 2.3, all the important technologies that were used over the course of the project, like *iptables*, *Snort* and the different chat bots that will be analyzed are introduced. After analyzing the relevant concepts and technologies, sections 2.4.1 to 2.4.4 analyze some related work and background.

Recently, the use of chat bots in different technical areas has become a focus of interest, particularly in cybersecurity. One specific application in this broad field, which will be addressed in this work, involves the use of chat bots to translate firewall and IDSs rules from natural language. Although nothing similar has been found in the literature, *SecBot* [8, 9] shows a similar inspiration, but follows a different approach, with similar, albeit broader, end goals. *CyberBench* and *CyberInstruct* [10] are two innovative tools that were designed to enhance the use of LLMs in the cybersecurity field, even though they do not tackle firewall and IDSs rules like this work does. Fine-tuning techniques are something that is being analyzed and used by chat bots in different areas, advocating in favor of this approach to try to increase their capacity in more specific areas, as is shown in section 2.4.3. At last, a broader search was made to understand how chat bots are being used in different security settings, which can be found in section 2.4.4.

2.2 Preliminary Concepts/Terminology

In this section, various concepts and technologies pertaining the areas of cybersecurity and AI, which are addressed in this dissertation, are discussed and explained.

2.2.1 Concepts in Cybersecurity

In this section, the type of systems to which rules will be written and the type of attacks behind those rules are defined as follows:

- Firewall – A firewall [11] is a network security system that monitors, filters and controls incoming and outgoing network traffic by following a defined set of security rules. Firewalls have been the front line of defense in network security for over 25 years. They normally place a barrier between a trusted internal network and untrusted external networks, like, for example, the Internet. Firewalls are used to protect a network or a host from malicious or unnecessary network traffic. Firewalls can be hardware, software or a mixture of both. There are three types of firewall topologies, them being:
  - Dual-homed gateway – The firewall is a single machine with two or more network cards, properly strengthened with firewall software and stripped of unnecessary services;
  - Screened host – The firewall consists of a router that connects the protected network to the outside and a gateway connected to the internal network by a single network card;
  - Screened subnet – The firewall consists of two routers, one that connects the protected network to a Demilitarized Zone (DMZ) and the other that connects the DMZ to the outside, and a gateway.

  Firewalls also have different operational models, which are described below:
  - Packet filter – Based on packet filtering, in this model security rules are defined and every single data packet is put through them to see if it passes and is allowed inside the network;
  - Circuit filter – The gateway establishes the Transmission Control Protocol (TCP) connections instead of any machine on the protected network and after verifying that the latter has the necessary permissions to do so, the remote connection is established;
  - Application filter - The gateway has several proxies running and mediates all communications between a client and a server, posing as the server to any computer within the protected network.
- Intrusion Detection System – An IDS [12] is a network security tool that monitors network traffic and devices for known malicious activity, suspicious activity or security

policy violations according to a defined set of rules. Contrary to firewalls, IDSs do not filter or control any traffic, they simply send out an alert when they find suspicious behaviour, making them unable to stop threats on their own, contrary to firewalls. IDS can be hardware or software. IDSs can be classified based on their operational features, them being:

  - Detection method – which can be signature based or anomaly based;
  - Source of events – which can be host based or network based;
  - Celerity of detection – which can be capable of operation at runtime, capable of operating in virtual real time or only operating during down time;
  - Reactivity - which can be active or passive;
  - Distributivity – which determines if the system acts alone (isolated) or not (cooperative).

- Denial of Service – Denial of Service (DoS) [13] is a type of cyberattack in which the attacker seeks to make the service, the computer, the network feed or the entire network unavailable to its legitimate users, often by interrupting the normal operation of one of the devices in between the user and the end service or by flooding the network to the point of being unresponsive. Distributed Denial of Service (DDoS) is a DoS attack that comes from many distributed sources. The most common DoS attacks work by flooding the target machine until its ordinary traffic is impossible to be processed due to the sheer volume of requests, resulting in ordinary users being locked out. Besides flooding attacks, there are some other types of DoS attacks like low and slow attacks, which rely on a small stream of very slow traffic targeting applications or server resources. Unlike the traditional brute-force attacks, low and slow attacks require very little bandwidth due to their volume being minimal and can be hard to mitigate, as the generated traffic can be quite similar to normal traffic in certain conditions, making some low and slow attacks go undetected for long periods of time, while they deny or slow the service for common users. The dataset that was created in the scope of this project contains rules for DoS attacks, namely the ones named Local Area Network Denial (LAND), Ping of Death, Smurf, Fraggle, SYN Flood, User Datagram Protocol (UDP) Flood, RST/FIN Flood, Slowloris, R-U-Dead-Yet (R.U.D.Y), Slow Read, Slow Post, Domain Name System (DNS) Amplification, Blacknurse, Low Orbit Ion Cannon (LOIC), High Orbit Ion Cannon (HOIC), Teardrop, WinNuke and Network Basic Input/Output System (NetBIOS) NT NULL Session Request;

- Buffer Overflow – Buffers are memory storage regions that temporarily hold data while it is being transferred from one location to another or being used by a program. A buffer overflow [14] (or buffer overrun) occurs when the volume of the stored data exceeds the predefined storage capacity of the memory buffer. When that happens, the program that is writing data to the buffer overwrites adjacent memory locations. Buffer overflows typically result from malformed inputs, be it intentional or not, or failure to allocate enough space for the buffer. If this process overwrites executable code, it can cause that same program to behave unpredictably and generate incorrect results, memory access errors, or crashes. Attackers exploit buffer overflow issues by overwriting the memory of an application. These attacks aim at changing the execution path of a program, triggering a response that damages files or exposing private information. For example, if an attacker introduces extra code, sending new instructions to the program, they can get full access to the system. Taking it a step further, if the attacker knows the memory layout of a program, they can intentionally feed input that the buffer cannot store, and overwrite areas that hold executable code, replacing it with malicious code. For example, if an attacker overwrites a pointer (object that points to another area in memory) and points it to an exploit payload, they can seize control of the system. The dataset that was created in the scope of this project contains rules for buffer overflow attacks, namely the ones named Squid Web Cache Communication Protocol (WCCP) Message Overflow, x86 Linux Samba Overflow, Internet Security Association and Key Management Protocol (ISAKMP) First Payload Certificate Request Length Overflow and Structured Query Language (SQL) Worm Propagation;

- MySQL/SQL Injection – SQL injection [15] is a common attack that uses malicious SQL code for backend database manipulation to access information that was not intended to be available to the attacker. This information may include a wide range of data, including private company data, user lists or customer details that should not be public. A successful attack may result in the unauthorized viewing of user lists, the loss of data by deletion of tables and, in certain cases, the attacker obtaining administrative access to a database, all of which are highly detrimental to a business. The dataset that was created in the scope of this project contains rules for SQL injection attacks, namely MySQL Root Login and MySQL Show Databases;

- Shellcode Exploits – A shellcode [16] is a small piece of code used as the payload in the exploitation of a software vulnerability. It is called “shellcode” because it typically starts a command shell, in the compromised machine, that the attacker has access to.

Shellcode is usually small and executable, which allows it to be used in as wide a variety of situations as possible. An exploit will commonly inject a shellcode into the target process before or at the same time as it exploits a vulnerability to acquire control over the program counter, which will be adjusted to point to the shellcode. After that, it will execute and perform its task. Injecting the shellcode is often done by storing the shellcode in data sent over the network to the vulnerable process. The dataset that was created in the scope of this project contains rules for shellcode exploit attacks, namely Secure Shell (SSH) Cyclic Redundancy Checksum (CRC)32 /bin/sh Overflow Exploit, Shellcode Scalable Processor Architecture (SPARC) setuid 0, Shellcode x86 setgid 0 and Shellcode x86 setuid 0;

- Remote Access Trojan – A *trojan* [17] is a malicious program that acts as a normal program, disguising itself as a legitimate software or piece of software generally. *Trojans* are spread through email or downloadable content. A Remote Access Trojan (RAT) [18] is a malware program that opens a secret communication tunnel that gives the attacker control of the device and the ability to access or steal data and deploy other software. RATs are spread like normal *trojans*, but the difference is, due to the control gained by the attacker, they can perform a wider range of malicious acts, making them more dangerous. The dataset that was created in the scope of this project contains rules for a RAT malware attack, namely for the one named SubSeven;

- Cross-Site Scripting – Cross-Site Scripting (XSS) [19] attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker sends malicious code using an unprotected web application, generally in the form of a browser side script, to a different end user. These attacks can occur anywhere a web application uses input from a user within the output generated and it is not validated or encoded. The browser of the victim has no way to know that the script should not be trusted, and will execute the script as it thinks it came from a trusted source. Also because of that, the malicious script can access any cookies, session tokens and other private information saved by the browser and used within that site. These scripts can even rewrite the content of an HyperText Markup Language (HTML) page. The dataset that was created in the scope of this project contains rules for an XSS attack.

2.2.2 Concepts in Artificial Intelligence

In this section, the definition of a chat bot and methods related to fine-tuning and to evaluate chat bot responses are explained as follows:

- Chat bot – A chat bot [20] is a computer program capable of simulating human conversation with an end user using conversational artificial intelligence techniques like Natural Language Processing (NLP) and LLMs, to understand what the user is asking of him and generate a response to him or her. Besides those, there are also rule-based chat bots, that follow a set of rules to respond to user queries, just like the one referenced in 2.4.1, and hybrid chat bots that combine both approaches. Their main goal is to provide users with apparently semantically correct information. They can be used for various services, like customer support, and are increasingly being used in various industries, including programming, healthcare and retail, to streamline operations, improve user satisfaction and reducing wait times and costs;

- Fine-tuning - Fine-tuning [21] is an approach commonly used in deep learning[1], in which a pre-trained model is trained on new data, normally to improve on a more complex task or to perform a new skill or task in which the fine-tuned model can use the knowledge the pre-trained model has without needing to start from scratch. In the specific case of LLMs, they undergo fine-tuning because they already possess an understanding of the language and can transform that in a response, meaning only the model has to to adapt by learning the data necessary to that specific task, because the general knowledge that it already has will serve as a foundation and will make the process a lot faster than developing a model from scratch;

- Parameter-Efficient Fine-Tuning - Fine-tuning large pre-trained models, such as LLMs, is a computationally intensive process that could require adjusting millions of parameters. The traditional approach, while effective, can be a bottleneck for adapting these models to specific domains due to the high computational cost and time requirements. Parameter-Efficient Fine-Tuning (PEFT) [22] methods were developed to allow the use of these larger models, using less resources while keeping the same performance. PEFT is a technique used to efficiently fine-tune LLMs on specific tasks without needing to update all the parameters of the model, therefore, being a more efficient alternative to full fine-tuning. In addition to being more efficient, other benefits of PEFT include:

[1] Deep learning is a specialized field within machine learning that utilizes deep neural networks, intricate architectures inspired by the human brain, to enable machines to learn and make intelligent decisions from vast amounts of data.

– Decreased computation and storage costs – by freezing the majority of the pre-trained parameters and focusing on training a minimal set of new ones, PEFT greatly reduces the computation resources needed;

– Better performance in low-data regimes – PEFT is particularly effective in scenarios with fewer data, and can achieve a superior performance and better generalization than traditional fine-tuning methods;

– Comparable performance with fewer trainable parameters – PEFT offers a cost-effective approach to fine-tuning while also reducing significantly the number of trainable parameters. This makes PEFT suitable for a wide range of applications, regardless of computational constraints.

There are various PEFT methods, like LoRA, which is used in this work and is explained in detail in the item below;

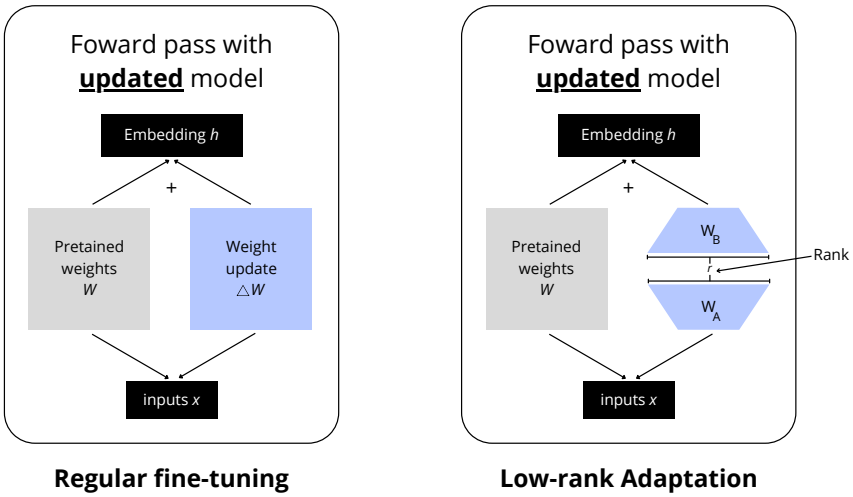

Figure 2.1: Scheme of the difference between LoRA and regular fine-tuning.

- LoRA – LoRA[23] is a PEFT method, which means that it shares its benefits. LoRA improves fine-tuning efficiency by representing weight updates as two smaller matrices through low-rank decomposition. These update matrices are trained to adapt to new data while minimizing the number of changes to the original model. The original weights remain frozen, and the final results are produced by combining the original and adapted weights.

  This approach offers a number of advantages:

- LoRA improves fine-tuning efficiency by greatly reducing the number of trainable parameters (by 10000 times);
- By freezing the original pre-trained weights, LoRA allows for the creation of multiple lightweight models for various downstream tasks;
- LoRA can be seamlessly integrated with other PEFT techniques to further optimize fine-tuning processes;
- LoRA does not introduce any additional inference latency, as the adapter weights can be merged with the base model.

A schematic showing how LoRA works can be seen in the figure 2.1. The diagram shows the difference between regular fine-tuning and LoRA. On the left is regular fine-tuning, where the weight update is saved separately in an array. What happens in LoRA is that the weight matrix is decomposed: instead of learning the whole matrix, LoRA learns two smaller matrices whose size depends on a rank, r. This rank is a number that allows many parameters to be saved, which makes training more efficient. For a more concrete example, imagine that the parameter matrix has dimensions of 5000 x 1000, which means that, in regular fine-tuning, we would have a matrix with five million parameters. If in LoRA the rank is 10, we would have two smaller matrices, one 10 x 1000 and the other 5000 x 10, a total of 60000 parameters, many fewer than in the other case;

- Quantized Models – Quantization[24] is a technique that reduces computational and memory costs of running inference by using low-precision data types, like 8-bit integers instead of the usual 32-bit float, to represent the weights and activations.

  By reducing the number of bits used to represent model weights and activations, many benefits can be achieved, in terms of memory storage, energy consumption and computational efficiency. Integer arithmetic can be performed much faster than floating-point arithmetic, allowing for accelerated operations like matrix multiplication. Additionally, this approach enables the deployment of models on embedded devices with limited hardware capabilities.

  In this study, the fine-tuned models are quantized models using Post-Training Quantization (PTQ) [25]. PTQ is a quantization technique that takes the base model and reduces the precision of the weights and activations of the same model, without requiring retraining. This method reduces the size and computational complexity of the

model, making it more efficient in resource constrained environments, while maintaining a pleasant performance. An alternative that would keep a better performance is Quantization-Aware Training (QAT) [26], but it would require the retraining of the base model, which would not be feasible, because the datasets for these models are not public and the time and computational costs of training a model with a dataset of that size would still be costly, despite the cost reduction brought by this method;

- Batch Size – Batch size [27] is a machine learning hyperparameter that defines the amount of samples the model goes through before updating the weights of the model. Batch size has an impact on training speed and memory consumption, with smaller batch sizes having faster training interactions, despite taking more time to converge, and using less memory, making it more friendly to less powerful machines;

- Gradient Accumulation Steps – Gradient accumulation [28] is a technique used in machine learning to train models with larger effective batch sizes without requiring excessive Graphics Processing Unit (GPU) memory. This technique is particularly useful when dealing with large datasets or complex models.

  Gradient accumulation works as following:

  1. Smaller batches – instead of using a big batch size, that might overwhelm the memory of the GPU, the data is divided into smaller batches;
  2. Gradient calculation – for each smaller batch, the model calculates the gradients, which indicates the direction and strength of the discrepancy between the predictions of the model and the real values;
  3. Gradient accumulation – these gradients are accumulated over multiple smaller batches. This means that the values are either added together or averaged, depending on the implementation;
  4. Weight update – after a specified number of smaller batches, the accumulated gradients are used to update the weights of the model.

  Gradient accumulation is a technique that effectively simulates a larger batch size by aggregating gradients from multiple smaller batches. This can improve generalization and stability. Moreover, it allows for training larger models or datasets on hardware with limited memory, making it a valuable tool;

- Learning Rate – The learning rate [29] is a machine learning hyperparameter that controls the step size at each iteration as the model moves towards the optimal solution,

e.g., moving towards a minimum of a loss function. This parameter can be thought of as the speed with which the model learns. The learning rate encapsulates often an important decision. This parameter determines the step size at each iteration. Too large a step can cause the minimum of the loss function to be exceeded; too small a step can cause the model to take too long to converge or can cause the model to get stuck in an unwanted local minimum. Therefore, one must be careful when choosing this value.

Deciding on a learning rate is important because:

  - Both a low value and a high value result in wasted time and wasted resources;
  - A small learning rate value means more time the model needs to train;
  - A high learning rate can result in a model that might not predict anything accurately.

A good learning rate could be the different between a model that does not learn anything from a model that presents state of the art results;

- Optimizer – Optimizers are algorithms that adjust the learnable parameters of the model, such as weights and biases, in order to minimize the loss function.

  Gradient Descent [30] is one of the most fundamental optimizers. This is an optimization algorithm based on a convex function that iteratively adjusts its parameters to minimize a given function to its local minimum. In each iteration, Gradient Descent reduces the loss function by moving in the direction opposite to that of the steepest ascent. The advantages of this algorithm are that it is easy to both understand and implement, although, because this method calculates the gradient for the entire dataset in one update, this can be a slow process, which also means that is computationally expensive.

  Stochastic Gradient Descent (SGD) [31] is a variant of Gradient Descent. This algorithm processes each training example individually during an epoch, updating the parameters of the model after each one. Since only one training example needs to be hold, this is easier to store in memory. The main advantages of SGD are the frequent updates of the model parameters, which can lead to convergence in less time and requires less memory since there is no need to store values of loss function. On the other hand, this algorithm can lead to noisy updates and high variance.

  Another popular optimizer is Adaptive Moment Estimation (Adam) [32], and it is also the one used in this project. This is a method that computes adaptive learning rates

for each parameter. It stores both the decaying average of the past gradients and the decaying average of the past squared gradients. Adam works as follows:

1. Momentum – the algorithm keeps track of an exponentially decaying average of past algorithms. This is helpful in order to smooth out the updates and to navigate noisy gradients;
2. Adaptive Learning Rate – Adam computes an exponentially decaying average of past squared gradients, that is used to scale the learning rate for each parameter. This allows for larger updates for less frequent features and smaller updates for more frequent ones;
3. Bias Correction – initially, the averages computed by Adam may be biased towards zero, but the algorithm incorporates a bias correction step to counteract this early bias.

The main advantages of the Adam optimizer are that the method is fast and converges rapidly and rectifies vanishing learning rate and high variance. However, it can be computationally costly;

- Weight Decay – Weight Decay [33] is a regularization technique that is commonly used to prevent overfitting [2]. It works like a regularizer, by adding a penalty term to the loss function to penalize larger weights.

  Some of the benefits of Weight Decay are listed bellow:

  - Reduces overfitting – by preventing the weights from being too large, this technique helps reducing the complexity of the model and preventing it from fitting to the training data;
  - Improves model stability – by preventing the weights from becoming too large, weight decay can help to stabilize the training process, which can lead to an improvement of the robustness;
  - Improves Generalization – a model with smaller weights is often more likely to generalize well to unseen data.

- Seed – Seeds [35] are random numbers or vectors that are used to initialize various processes, that ensure that the results are not deterministic and exhibit desirable properties. Seeds are important because setting a specific seed ensures that the same sequence

[2] Overfitting [34] occurs when a machine learning model provides accurate predictions for training data, but not for new data.

of random numbers is generated each time the code is run, making the results reproducible, and, by changing the seed, different results can be obtained and compared.

- ROUGE – Recall-Oriented Understudy for Gisting Evaluation (ROUGE) [36] is an automatic evaluation package of summaries and commonly used to evaluate machine translation in NLP. ROUGE has five evaluations metrics available:
  - ROUGE-N evaluates the overlap of n-grams[3] between the expected and outputted answers. For example, ROUGE-1 specifically compares the common words between the expected and outputted answers, while ROUGE-2 compares a group of two words between the expected and outputted answers;
  - ROUGE-L compares the length of the Longest Common Sequence (LCS) of the outputted answer with the length of the expected answer, but does not take into consideration if the LCS is consecutive or not;
  - ROUGE-W compares the length of the LCS of the outputted answer with the length of the expected answer, just like ROUGE-L, but gives different weights if that sequence is consecutive or not;
  - ROUGE-S compares how many skip-bigrams, that is, any pair of words in the sentence in order, allowing arbitrary gaps, are common between the expected and outputted answer;
  - ROUGE-SU is an extension of ROUGE-S that takes into account unigram occurrences, that means, single words co-occurrences between the expected and outputted answers.

2.3 Important Technologies

This section introduces important technologies for the development of this work. The concepts of *iptables*, *Snort* and chat bots are defined, as well as the chat bots used in this dissertation.

2.3.1 *Iptables*

Iptables [37] is a C-written software that allows a system administrator to configure the Internet Protocol (IP) packet filter rules of the Linux kernel firewall. In the Linux operating system, the firewalling is taken care of by netfilter [38], which is a kernel module that

[3] An n-gram is a contiguous sequence of n adjacent items in particular order. These symbols may be n adjacent letters, syllables or words.

decides what packets are allowed in and out and *iptables* acts as the interface to netfilter. *Iptables* rules can be used to define if certain packets are accepted, dropped, rejected, transform and forward those packets to other softwares if they meet certain criteria, like protocol, source and destination address and port number.

2.3.2 *Snort*

Snort [39] is an open-source and lightweight network IDSs that has the ability to perform real-time traffic analysis and packet logging on Internet protocol networks. *Snort* uses a series of rules to identify potential malicious network and generate alerts for users when those rules are triggered. Unlike *iptables*, *Snort* can only alert when certain packets meet certain criteria, instead of defining what happens to that packet, but in terms of criteria, *Snort* has a lot more options when devising rules, like being able to analyze the content of a packet.

2.3.3 Chat bots

A set of chat bots were selected to be used in this study, each of them for a different reason. These chat bots were *ChatGenerative Pre-trained Transformer (GPT)* [40], *Gemini 1.5 Pro* [41], *Microsoft 365 Copilot* [42], *LLaMA* [43], *Mistral 7B* [44], and *GPT4All Falcon* [45], *Wizard v1.1* [46] and *Nous-Hermes* [47]. As state-of-the-art chat bots, *ChatGPT* and *LLaMA* were chosen. The *ChatGPT* version chosen was *ChatGPT* 3.5 as it was the currently publicly available free version of *ChatGPT* at the time of testing. The *LLaMA* version chosen was *LLaMA 2 7B* due to hardware limitations. *Mistral 7B* was selected as it stands out as a model that outperforms *LLaMA 2 13B* in all the benchmarks it was evaluated and *LLaMA 1 34B* in code generation, among others. As less Random Access Memory (RAM)-intensive options, which is a potential bottleneck when running and fine-tuning chat bots, *Falcon*, *Wizard v1.1* and *Nous-Hermes* were chosen, with the first requiring 8GB of RAM and the other two 16GB. These three models achieve some of the best results when taking into account the memory constraint and ignoring quantized models, as can be seen in the benchmarks published by GPT4All [47]. *Gemini 1.5 Pro* and *Microsoft 365 Copilot* were added at a latter date, as they were made public after the initial study was made, but are renowned as state-of-the-art chat bots, so they were analysed as well.

2.3.3.1 ChatGPT 3.5

ChatGPT 3.5 is an instruction based model known for being one of the best chat bots on the market and it was trained by OpenAI [48]. ChatGPT is a sibling model to *InstructGPT*, which is trained to follow an instruction in a prompt and provide a detailed response. *ChatGPT* is also able to answer followup questions, admit its mistakes, challenge incorrect premises, and reject inappropriate requests. *GPT-4* can solve difficult problems with greater accuracy, thanks to its broader general knowledge and problem solving abilities and surpasses his previous version in its advanced reasoning capabilities.

2.3.3.2 LLaMA

LLaMA [43] is an instruction based model known for being one of the most powerful chat bots available and it was trained by Meta AI [49]. *LLaMA* was designed to help researchers advance their work in this field, with a smaller, but more performant model, which is desirable because it requires far less computing power and resources to test new approaches, validate the work of others and explore new use cases. *LLaMA 2* was trained on 40% more data and has double context length.

2.3.3.3 Mistral 7B

Mistral 7B [44] is a 7-billion-parameter language model engineered for superior performance and efficiency. This model leverages architecture choices like grouped-query attention for faster inference and sliding window attention to effectively handle sequences of arbitrary length with a reduced inference cost.

2.3.3.4 GPT4All Falcon

GPT4All Falcon [45] is an instruction based model known for fast responses that was trained by the Technology Innovation Institute and fine-tuned by Nomic AI [50]. It was trained over a massive curated corpus of assistant interactions including word problems, multi-turn dialogue, code, poems, songs and stories. This model is a fine-tuned version of the Falcon on assistant style interaction data.

2.3.3.5 Wizard v1.1

Wizard v1.1 [46] is an instruction based model known for very long responses. It was trained by Microsoft [51] and Peking University [52] and it was fine-tuned with a small amount of

high quality data. Wizard is a LLMs trained using a new method, called Evol-Instruct [53], on complex instruction data. Wizard outperforms similar LLaMA-based LLMs and the version 1.1 was released with significantly improved performance.

2.3.3.6 Hermes

Hermes [47] is an instruction based model known for long responses that was trained by Nous Research and was curated with 300000 uncensored instructions. The model was trained almost entirely on synthetic GPT-4 outputs. It is suitable for a wide range of language tasks, from generating creative text to understanding and following complex instructions.

2.3.3.7 Gemini 1.5 Pro

Gemini 1.5 Pro [41] is an advanced LLM developed by Google DeepMind [54]. The model surpasses state-of-the-art performance across a broad set of benchmark. *Gemini 1.5 Pro* introduces a context window of up to two million tokens, which is the longest context window of any large scale foundation model yet, allowing it to handle large inputs across various tasks, such as code generation, long-form document analysis and more.

2.3.3.8 Microsoft 365 Copilot

Microsoft 365 Copilot [42] is an AI-powered tool developed by Microsoft, that can act as a chat bot as well, and is integrated across all Microsoft 365 applications. It was developed with the objective of helping automate repetitive tasks and enhance productivity. It harnesses LLMs, which include pre-trained models like *GPT-4*, that are designed to excel in these types of tasks.

2.4 Related Works

In this section, various works related to the work done in this dissertation, from the use of chat bots in cibersecurity, to fine-tuning, to the use of chat bots in other security settings are presented and discussed.

2.4.1 SecBot

SecBot [8, 9] is a chat bot that seeks to help businesses by providing support in cybersecurity planning and management. It identifies cyberattacks based on related symptoms and

indicates solutions and configurations taking into account business needs, providing information relevant to the investments and risks that come with certain cybersecurity related decisions. *SecBot* asks for specific business information, such as budget, and also values or settings that it needs for its analysis. After obtaining all the information it needs, the chat bot recommends a solution based on the request, the information given by the user and the analysis made. This recommendation is made using pre-written rules, which extract the necessary information and return the solution. An example of the way *SecBot* processes input and returns an output is the following:

input: *"I have iptables installed and I want to protect my network against an ICMP flood"*

entities_extraction *{*
"intent":solution_configuration
"solution":IPtables
"operator":protect
"target":network
"attack_name":ICMP flood
}

custom_action: *"find_configuration(solution, action, target, attack_name)"*

output: *"The command for your configuration request is:* `iptables -t mangle -A PREROUTING -p icmp -j DROP`"

In this example, it can be seen *Secbot* extracting the information it needs from the "input" to the `entities_extraction` field. After that, it invokes the `custom_action` with the data extracted as parameters and gives an "output" as a response to the request.
SecBot has a similar purpose to this work, that being recommending security setting in the cybersecurity field, but differs in being business oriented and not using fine-tuning to achieve this, but pre-determined rules.

2.4.2 CyberBench and CyberInstruct

Liu *et al.* [10] found that LLMs had the potential to be used for specific cybersecurity tasks, but according to the research conducted, this has not yet been explored. A benchmark was proposed for evaluating LLMs within the cybersecurity domain. This tool is *CyberBench*. *CyberBench* is a multi-task benchmark that was designed to evaluate the performance and efficiency of LLMs in cybersecurity tasks in English. Ten datasets were selected and divided into

four categories: named-entity recognition, summarization, multiple-choice and text classification. Several evaluation methods were used, including ROUGE-1, ROUGE-2 and ROUGE-L. The second tool developed was *CyberInstruct*. *CyberInstruct* is a family of generative, fine-tuned LLMs that has been designed to deal specifically with the unique challenges and inherent requirements of the cybersecurity domain. These tools specialize in various cybersecurity tasks, according to the content of each dataset used to train them. Some examples are the identification and classification of cyberentities, such as exploits and vulnerabilities, phishing email and website detection, log anomaly detection, threat analysis and malware classification, among others. Given that the performance is good and that it is adaptable, the base model (or foundation) used to be fine-tuned in the context of cybersecurity is Llama-2. The fine-tuning process involves taking advantage of *CyberBench* datasets to create a specialized model with improved cybersecurity capabilities. A technique called instruction-tuning is also used, which involves incorporating explicit instructions into the input of the model during the fine-tuning process, which leads to the model generating a more accurate and more relevant output. PEFT is also used, using Quantization and LoRA (QLoRA) when *CyberInstruct* is fine-tuned. The results obtained were promising and show that this is an approach for adapting LLMs in the cybersecurity domain.

2.4.3 Fine-tuning Chat bots

Fine-tuning is a technique that provides a dataset with specific prompts and example answers, with the aim of improving the capabilities of a model in that specific knowledge niche. The work described in [55] uses ChatGPT-4 to generate responses to 52 000 prompts and then repeats the process in Chinese, translating the prompts and making ChatGPT-4 respond to them in Chinese as well. This process was carried out to generate the datasets that would be used to fine-tune the *LLaMA 7B* model. The fine-tuned version of the *LLaMA 7B* model showed a considerable improvement in performance, validating the use of fine-tuning.The authors of [56] used reinforcement learning through human feedback to train a therapy chat bot, using interactions between therapists and clients to fine-tune a model. No significant differences were noted between the models before and after fine-tuning, something the authors attributed to the small amount of data used, something that should be explored in future work. These works show that, while fine-tuning is a valid approach to improving a model, it is not a guaranteed success, requiring a well-constructed dataset to achieve the desired result.

2.4.4 Chat bots in Other Security Settings

Some of the topics currently discussed when talking about the use of chat bots in cybersecurity-specific areas are concerns about data protection, privacy and other social aspects. Ethical considerations, such as not sharing personal data without permission and the need to avoid bias, reducing the risk of adverse attacks, maintaining data integrity and building user trust are some of the reasons why data protection and privacy are important to take into account when using chat bots [57].

Hasal *et al.* [58] also shows concern about factors that can influence the behavior of a model, such as privacy risks and data leakage, more precisely the unintentional sharing of confidential information during data transmission and data poisoning. The use of data anonymization, privacy-aware machine learning algorithms, adversarial training, robust training, like the use of contradictory examples to improve the model and testing how the model handles these examples that will be seen as unexpected, data verification, limiting the rate of requests and the use of encryption are some of the recommendations used to mitigate privacy exploits and strengthen data protection. A framework was also created in order to test and analyze how vulnerable a chat bot is to hackers [59] and, to this end, it checked whether chat bots were vulnerable to XSS and SQL injections and the results obtained were positive, proving that common attacks would not be effective against the chat bot tested.

The benefits, challenges, risks and consequences of integrating generative AI tools into traditional pentesting frameworks were also studied in a work that approached using *ChatGPT* in pentesting [60]. The benefits highlighted were improved efficiency and continuous learning, which are certainly positive, but there were a lot of risks and consequences like over-reliance and ethical, legal and bias concerns. The authors of [61] also explored pentesting using *ChatGPT*, focusing specifically on the reconnaissance phase, drawing the conclusion that it is a valuable tool that provides useful information. It is also worth to mention that this study also concluded that the way a prompt is written affects the response given by the chat bot, something that should be taken into account when using one or when creating a set of prompts to present to a chat bot, like it will be done in this work, by putting together a dataset to fine-tune a model. The capability to generate malicious code or messages that produced attacks against systems or users, such as phishing emails, was investigated by Qammar *et al.* [62], using *ChatGPT*. It was concluded that *ChatGPT* succeeds in creating this type of content, emphasizing the capability to produce undetectable zero-day attacks.

2.5 Conclusion

This chapter explored the state of the art surrounding the main subjects of this dissertation, which includes the literature related to the area of this work, all the preliminary concepts that will help understand this project better and the important technologies that were used during this study. The specialized literature highlights the potential of using chat bots in various tasks that have similar contexts to this work and their fine-tuning, but also some of the risks associated with their use, like privacy, data protection and over-reliance concerns, of which users should always be wary of. This chapter introduced what was previously researched and helped get a baseline for this work and gave a preliminary understanding to everything that will be discussed and applied in latter chapters.

Chapter 3

Method

3.1 Introduction

This chapter introduces the method used during this study. It explains the data collected, in section 3.2, which comprises the identification of cyberattacks and rules, the writing of the prompts and the choice of the chat bots. After that, the fine-tuning will be addressed, explaining the overall fine-tuning method in section 3.3, and the four fine-tuning approaches in greater detail from section 3.4 onwards.

3.2 Data Collection

At the start of this work, there was an in-depth research that aimed to find an existing dataset that included attacks, or prompts of attacks, and potential answers comprising the corresponding *iptables* and *Snort* rules, but that search was unsuccessful and it is believed such a dataset does not exist. Inevitably, it was necessary to create a dataset and, that meant collecting data through the identification of a set of cyberattacks that could be avoided by *iptables*, *Snort* or both, and the respective rules that could prevent them both *correctly* and *efficiently*. The rules need to be *correct* so they can prevent the attack, and *efficient*, so they avoid blocking legitimate traffic. This collection of data will comprise the dataset that will serve as a foundation for the evaluation of the rule-writing capabilities of the various chat bot models and for the datasets that will be used to fine-tune a model. The analysis of the generated rules will help assess the capabilities of these chat bots before and after fine-tuning. This dataset, which is a contribution of this work, is made available in the GitHub repository in `https://github.com/BernardoLouro/rgft`. Paired with the dataset, there are also instructions to replicate the experiments conducted during this study in the said repository.

3.2.1 Identifying Cyberattacks and Rules

Thirty cyberattacks were collected, along with the corresponding *efficient* and *correct iptables* and *Snort* rules. There was some variety of identified attacks, despite most of them being DoS or DDoS attacks, which included low and slow attacks like R.U.D.Y and floods

like TCP flooding. Besides those, other attacks included buffer and integer overflow exploits, MySQL/SQL injections, shellcode exploits, a RAT and XSS. For an extensive explanation about each of these attacks, please refer to section 2.2.1. Table 3.1 displays the distribution of the types of attacks.

Table 3.1: Distribution of Attacks.

Type		Number of Attacks
DoS	Floods	9
	Low and Slow	4
	Other	5
Overflow Exploits		4
SQL Injection		2
Shellcode Exploits		4
RAT		1
XSS		1

3.2.2 Writing Prompts for each Cyberattack

As one of the works mentioned in section 2.4.4, the way a prompt is worded can affect the answer given by the chat bots, so there was a need to understand how that would affect results. Taking that into account, three different prompts were written for each cyberattack and each was intended to reflect the way people with different knowledge levels would ask a chat bot how to create a rule for a cyberattack.

These three prompts could be divided into the expected prompt from (i) a person with no knowledge on the subject, (ii) a person with technical knowledge to understand the attack, but no knowledge on how to prevent it and (iii), a person with the technical knowledge who knows how to prevent it, respectively. To simulate that, the first prompt only asked for rules provided by the name of the attack, to represent a user that only searched for the most common attacks, without giving any insights on how they can be perpetrated, or how the attacks are prevented. The second prompt explains the technicalities of the attack, but does not supply any information on how preventive measures can be embedded into the rule. The third prompt clearly explains what the rule should do to avoid the attack. As an example, the following represents the three different prompts for the Ping of Death attack:

- First Prompt (user with no knowledge) - *Write an iptables rule to avoid Ping of Death attacks*;
- Second Prompt (user with limited knowledge) - *Write an iptables rule that avoids an*

attack in which the attacker sends a malformed ping packet that exceeds the maximum IPv4 packet size;

- Third Prompt (user with knowledge) - *Write an iptables rule that drops Internet Control Message Protocol (ICMP) packets with a length greater than 65,535 bytes.*

3.2.3 Choosing Chat Bots

After finishing writing the prompts, they were presented to the set of selected chat bots, as a way to evaluate their capabilities to write *iptables* and *Snort* rules, before fine-tuning. The verification of the responses was manually performed, to not only verify if the solution was correct, but also to check for the technical prowess of the chat bots to write the syntax of *iptables* and *Snort* rules. The chat bots were chosen for different reasons, be it their status as state of the art chat bots or because they are the best performing in memory constraint environments. For an extensive explanation about the choices of the chat bots, please refer to section 2.3.3. The results of this evaluation are presented in section 4.2.

3.3 Fine-tuning Method

Following the analysis of the capabilities of currently available chat bots, it was time to fine-tune them. The method of choice was supervised learning and, because of time and hardware constraints, Unsloth [63] was used, which reduced training time and the use of Video Random Access Memory (VRAM) by half, due to the use of quantized [24] models and state-of-the-art PEFT methods [22], *e.g.* LoRA [23], which were explained in greater detail in section 2.2. With the required computational resources being lower than fully fine-tuning a model, this method is more accessible to users, which could possibly motivate the replication of this method or their use for another projects.

During this work four different fine-tuning approaches were explored, each with a different objective that was explored through the way the dataset was structured, which will be explained with greater detail in the next section. When it comes to the parameters used during the fine-tuning, consistency was a highly priority, so the parameters of all the models were kept equal, besides one exception that will be justified when mentioned. Table 3.2 displays the values of the most pertinent parameters. This set of parameters represents a balance between performance and resource/time consumption.

Table 3.2: Selected fine-tuning parameters.

Parameter	Value
Batch Size	2
Gradient Accumulation Steps	4
Steps	600
Learning Rate	2×10^{-4}
Optimizer	Adam
Weight Decay	0.01
Seed	3407

3.4 Fine-tuning Approaches

As mentioned before, four approaches were developed and tested as a way to evaluate the capability of the models to generate rules.

Approaches #1 and #4 saw the model being trained by the prompts that were previously created to test them before fine-tuning. The structuring was kept consistent throughout the entire dataset, so the testing saw an adjustment of the structure of the question to evaluate if it could still answer correctly. The dataset for these approaches consisted of 126 prompt-response pairs.

Approach #2 saw the model being trained with a dataset of two different types of prompt-response pairs, one that asked for a description of a name of an attack and the other asking for a *iptables* or *Snort* rule for that same description. This approach aimed to analyze if the model could link the attack and the rule through the common information (description). The dataset for approach #2 consisted of 84 prompt-response pairs.

Approach #3 saw the model being trained with simple *iptables* and *Snort* rules to evaluate if the model could infer rules that are more elaborate, by putting together the simpler ones. The dataset for approach #3 consisted of 81 prompt-response pairs. Examples of the prompt-response pairs for all the models mentioned in this section are included below.

As mentioned before, the developed datasets are also available on the GitHub of the project and a thorough description and examples will be presented in the following sections, with the results following in the next chapter.

3.4.1 Approach #1

Approach #1 was thought with two evaluations in mind. The first being the comparison between two models and the second is if the model is capable of answering the question correctly when the structure of the question is altered.

The two models chosen were *Mistral 7B* [44] and *LLaMA 2 7B* [64] due to their different performances before fine-tuning, the first being the best performing model without pricing associated to fine-tuning and the latter being one of, if not the worst performing model, results which will be addressed in greater detail in the next chapter. That difference should be able to highlight if a difference between the baseline state of a model affects the performance of the fine-tuned model.
As for the changing in the structure of the prompts, the dataset used for training always followed the following structure:

Write a(n) (iptables/snort) rule that ...

Alternatively, the testing prompts followed slightly altered structures, namely:

How to ... with (iptables/snort)?,
What (iptables/snort) rule ...?, or
Develop a(n) (iptables/snort) rule to ....

This variation in structure aims to evaluate the capability to answer a question correctly even when it is not asked the exact same way.

3.4.2 Approach #2

Approach #2 was thought as a way to evaluate if the model could take information given to him by two different prompts and link them. In this case, the knowledge that is supposed to be linked is the name of the attack and the rule, while the knowledge that is shared between the two rules is the description of the attack. So the first prompt-response asks for a description of an attack through the name, while the second prompt-response asks for a rule through the same description given.
An example of each of the prompt-response pairs is as follows:

- First Prompt-Response Pair Example:

  Prompt: *What is a Local Area Network Denial (LAND) attack?*

  Response: *It is an attack in which the attacker sets the source and destination information of a TCP segment to be the same.*

- Second Prompt-Response Pair Example:

  Prompt: *Write an iptable rule that avoids an attack in which the attacker sets the source and destination information of a TCP segment to be the same.*

Response: `iptables -A INPUT -s <your_network_IP> -d <your_network_IP> -j DROP`.

3.4.3 Approach #3

Approach #3 was thought as a way to evaluate if the model could learn *iptables* and *Snort* rule syntax through simple and generic rules and apply that knowledge to generate complex rules by putting together different rules, while keeping the syntax correct.
An example of what could be classified as a simple and generic rule and a complex rule that generated from those simple rules are as follows:

Prompt: *Write a snort rule to alert tcp packets from the port 43267.*

Response: `alert tcp any 43267 -> any any (msg:'packet from port 43267';)`

Prompt: *Write a snort rule to alert icmp packets to a port.*

Response: `alert icmp any any -> any <port_number> (msg:'packet to port <port_number>';)`

Prompt: *Write a snort rule to alert tcp packets with a flow established to server.*

Response: `alert tcp any any -> any any (flow: established,to_server; msg:'packets with established connection and flow to server';)`

Prompt: *Write a snort rule to alert tcp packets with the content 'virus'.*

Response: `alert tcp any any -> any any (content: 'virus'; msg:'packet with content 'virus'';)`.

Complex Prompt: *Write a snort rule to alert tcp packets to port 3306, with the flow to server and established and the content "|0A 00 00 01 85 04 00 00 80|root|00|".*

Complex Response: `alert tcp any any -> any 3306 (msg:'packet to port 3306, with established connection and flow to server and the content "|0A 00 00 01 85 04 00 00 80|root|00|"'; flow:to_server,established; content:'|0A 00 00 01 85 04 00 00 80|root|00|';)`.

3.4.4 Approach #4

Approach #4 was thought as a way to evaluate if the model was capable of identifying the rule if it was presented to it, and, motivated by [10], it was done by asking multiple choice questions to the chat bot. The model used was the same one that was used on 3.4.1 (Approach #1) and it was able to answer the multiple choice questions without further training focused on teaching that, by using the following prompt style:

Which one of the following options is true for this sentence:

Which option gives a(n) (iptable/snort) rule to ...:

Option A: ...

⋮

Option D: ...

Response:

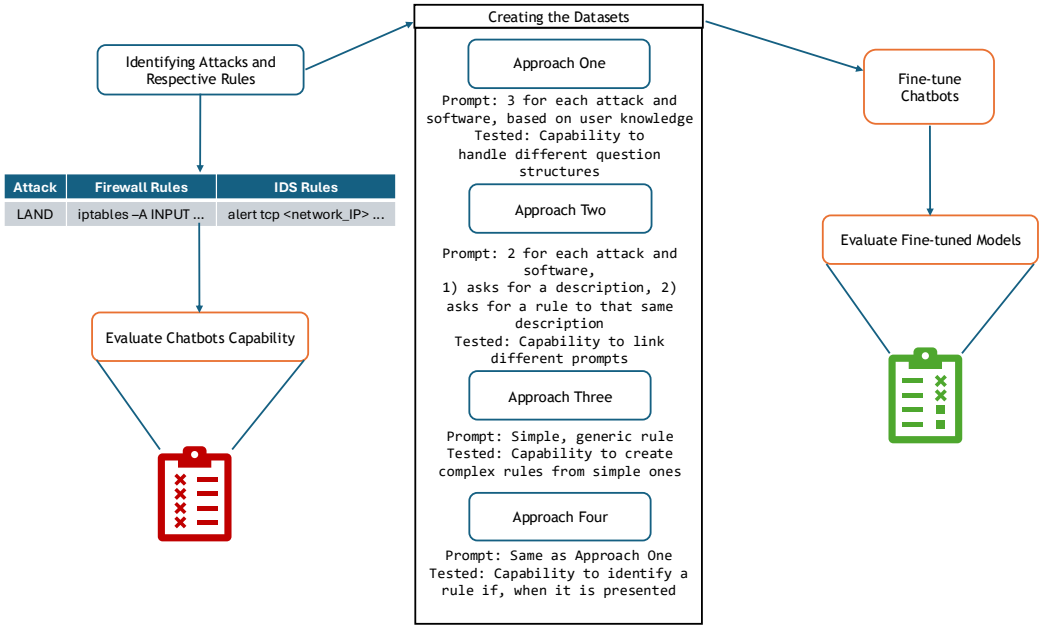

Figure 3.3: Illustration of the workflow followed.

3.5 Workflow

Figure 3.3 illustrates the workflow that was followed through this chapter, starting from identifying the attacks and rules and the negative evaluation of currently available chat bots. This figure highlights the reason behind each approach, by presenting how the prompts were designed for training and what was tested in each one of them. Approach #1 saw 3 prompts

designed for each attack and software, each to be written by people with different levels of knowledge and tested the way the chat bot handled a different structure to the same question. Approach #2 saw 2 prompts for each attack and software, with the first asking for a description and the second asking for a rule to that same description and tested the capability to link these two prompts by the knowledge linking the two prompts. Approach #3 saw simple and generic rules as prompts to test if the chat bot was capable of generating complex rules by combining multiple simple ones. At last, approach #4 used the same training dataset as approach #1 and tested if the chat bot was able to identify and discriminate if a rule was correct when presented in the form of multiple choices.

3.6 Conclusion

This section outlines the different steps required to complete this study, giving a better idea of what had to be done to complete it, a better understanding of what each decision made during the project was trying to accomplish and providing guidance for anyone looking to replicate this method. The next chapter will present the various results obtained from following the methodology in this chapter.

Chapter 4

Results

4.1 Introduction

This chapter introduces the results obtained during this work, starting with the preliminary evaluation, in section 4.2, which will give insight into the capability of writing *iptables* and *Snort* rules of current chat bots and a deeper analysis into the data collected. Following that, there will be the evaluation of the fine-tuning results, in section 4.4, in which the results of each approach are discussed in great detail, some comparisons with the preliminary evaluation are done and the ROUGE metric is used to evaluate the results.

4.2 Preliminary Evaluation

This section presents and discusses the results of the preliminary and manual evaluation carried out on the chat bots, regarding their ability to write rules with their base knowledge, without any extra intervention or fine-tuning. This evaluation was conducted by evaluating the success rate of the chat bots in the generation of rules through the prompts offered. The metric used was the success rate, which was calculated through a human analysis of the answers given by the chat bots and considering only the *correct* and *efficient* answers, for example the rule "`iptables -A INPUT -dport 53 -j DROP`" will avoid any attack coming to the DNS port, but it will also drop all the legit traffic, so it is technically *correct*, but not *efficient*. On the other hand, the rule "`iptables -A -p tcp --dport 53 -m connlimit --connlimit-above 10 -j REJECT`" will avoid a DNS Amplification attack and not drop any legitimate traffic, making it a *correct* and *efficient* rule. The best results belonged to *Microsoft 365 Copilot*, obtaining a success rate of 48%, that being 60 out of 126 correct answers, which was closely followed by *ChatGPT 3.5*, which got a success rate of 42%, with 53 out of 126 correct answers. Following them was *Gemini 1.5 Pro*, which got 34 out of 126 correct answers, or 27%. These three chat bots stood out positively as they were by far the ones with the highest success rates and despite the relatively low percentage value, most responses were close to the right answer, just not perfect. One example of this is when handling floods, the rule written was tracking the amount of packets by the IP address of the source, but the cor-

rect way to handle it would be to track the packets by the IP address of the destination, so botnets can be avoided, as can be observed in the rule for a UDP Flood: "`alert udp any any -> any any (flow:stateless; detection_filter:track by_src, count 50, seconds 10; msg: 'UDP Flood Detected';)`", for which the correct answer would be "`alert udp any any -> any any (flow:stateless; detection_filter:track by_dst, count 50, seconds 10; msg: 'UDP Flood Detected';)`". On the other side *LLaMA 2 7B* failed all 126 questions, revealing close to zero knowledge of of *iptables* and *Snort*. *Mistral 7B* had a better showing, displaying a stronger knowledge of *iptables* and *Snort* and getting a total of five out of the 126 questions. The three memory constrained models were not as successful as expected, with *GPT4All Falcon* getting only one, *Nous-Hermes* two and *Wizard v1.1* zero, with them all showing little knowledge of *iptables* and close to no knowledge when it came to *Snort*. The superior results obtained by *Microsoft 365 Copilot*, *ChatGPT* and *Gemini 1.5 Pro* make some sense, given how larger the datasets with which they were trained are compared with the other models, with none of them receiving any training focused on the topic tested in this work.

These results showcase how chat bots can output erroneous information, at least when the topic is related to cybersecurity tools rules. With cybersecurity being an area of focus due to the problems that attacks can bring, the use of chat bots and the potential consequences of using, for example, malformed rules, should lead to greater awareness, and this analysis shows that these models do not provide a high degree of trust for this purpose. Table 4.1 and Figure 4.2 summarize the achieved results from this systematic study of currently available chat bots, which will be approached in greater depth in the next section.

Table 4.1: Baseline performance comparison between different chat bots for generating *iptables* and *Snort* rules.

Model	Correct Answers	Success Rate
Microsoft 365 Copilot	60 out of 126	48%
ChatGPT 3.5	53 out of 126	42%
Gemini 1.5 Pro	34 out of 126	27%
LLaMA 2 7B	0 out of 126	0%
Mistral 7B	5 out of 126	4%
GPT4All Falcon	1 out of 126	<1%
Nous-Hermes	2 out of 126	≃2%
Wizard	0 out of 126	0%

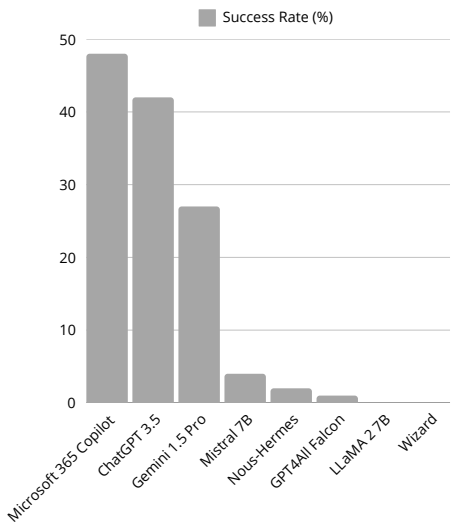

Figure 4.2: Baseline performance comparison between different chat bots for generating *iptables* and *Snort* rules.

4.3 Analysis of Data Collected

This section explores the achieved results and discusses some conclusions that were noticed after further analysis of the outputs produced by the chat bots before fine-tuning, like the difference in results for different types of prompts and the difference in results for *iptables* and *Snort*.

4.3.1 Results for Different Types of Prompts

The third type of prompt stood out as the one that obtained the best success rate out of all three prompts, which could be explained by the fact that it gives the answer to the problem instead of giving a problem and asking for a solution, so the chat bot only has to have the syntax knowledge to be able to give the rule. To put it into perspective, 37 out of 53 correct *ChatGPT 3.5* prompts were the third type, 22 out of 34 from *Gemini 1.5 Pro*, 35 out of 60 from *Microsoft 365 Copilot*, four out of five from *Mistral 7B*, one of the two from *Nous-Hermes* and the only one the *GPT4All Falcon* got right were all third type prompts. Those third type prompts added up to 65% of the correct answers. These results show that if the knowledge of the syntax used by *iptables* and *Snort* is known by the chat bots, their provided solutions, when given what they have to do in natural language, can result in a *correct* and *efficient* rule, at least in a much more efficient way than the other two methods tested.

4.3.2 Results for *Iptables* and *Snort*

As previously mentioned, one thing that stood out when analyzing the outputs given by the chat bots was the difference between the capability of the chat bots to write *iptables* rules as opposed to *Snort* rules. The results also support these observations with *ChatGPT 3.5* getting approximately 58% of the total *iptables* rules generated, while only 33% of *Snort rules*, *Mistral 7B* getting three out of the five correct and Nous-Hermes and *GPT4All Falcon* only achieving correct responses when outputting *iptables* rules. There were exceptions, namely *Gemini 1.5 Pro* and *Microsoft 365 Copilot*, with *Gemini 1.5 Pro* getting basically the same percentage for *Snort* and *iptables* rules, with 26% and 29%, respectively, and *Microsoft 365 Copilot* having *Snort* rules outperforming *iptables* rules obtaining 52% and 40%, respectively.

4.4 Fine-tuning Evaluation

In this section, the results of the fine-tuning are discussed and the different approaches are discussed in great detail.

4.4.1 Approach #1

With the different starting point from both models, it was thought that, to have a fair comparison, both models should be fine-tuned until their loss converged[1]. For that to happen, the *LLaMA 2 7B* model needed twice as many steps, being the exception mentioned before that did not fully abide by the parameters listed in Table 3.2. The results after fine-tuning stood out as the difference between the models aggravated, with *Mistral 7B* getting 89% of responses right, while the *LLaMA 2 7B* model continued having a 0% success rate. These results do not tell the whole story, as the model really improved and demonstrated much more syntax knowledge, coming closer to the right answer more often than not, though never outputting completely *correct* and *efficient* rules. After these poor results, *LLaMA 2 7B* was not considered for further analysis.

As has been emphasised throughout this work, the different choice of words and phrasing can have a big impact on the responses of a chat bot, as was already mentioned by *Temara et al.* [61]. A deeper analysis of the missed responses highlighted that again, as from the unsuccessful responses of the fine-tuned model, which were 14 misses, of those 13 were from

[1]Loss function convergence is a concept that refers to the process by which the loss function gradually decreases as the model is trained and approaches an optimal solution.

first type prompts, while the other was from a second type. The successful third prompt, became even more successful after fine-tuning, by having the chat bot not failing a single output when presented that type of prompt. The individual results for each prompt in this approach and the distribution of correct and incorrect answers sorted by type of prompt and software can be found in table 4.3 and figure 4.4.

Table 4.3: Individual result for each prompt in Approach #1. The columns are divided by the software and type of prompt. The checkmark means correct answer while the x means wrong answer.

Attack	**Result**					
	Iptables			**Snort**		
	1	**2**	**3**	**1**	**2**	**3**
LAND	✓	✓	✓	✓	✓	✓
Ping of Death	✓	✓	✓	✓	✓	✓
Smurf	✓	✓	✓	✓	✓	✓
Fraggle	x	✓	✓	✓	✓	✓
SYN Flood	✓	✓	✓	x	✓	✓
UDP Flood	✓	✓	✓	✓	✓	✓
RST/FIN Flood	x	✓	✓	✓	✓	✓
Slowloris	✓	✓	✓	✓	✓	✓
R.U.D.Y	✓	x	✓	-	-	-
Slow Read	x	✓	✓	-	-	-
Slow POST	x	✓	✓	-	-	-
DNS Amplification	✓	✓	✓	x	✓	✓
BlackNurse	x	✓	✓	x	✓	✓
LOIC	x	✓	✓	✓	✓	✓
HOIC	x	✓	✓	✓	✓	✓
SubSeven	-	-	-	✓	✓	✓
Teardrop	-	-	-	x	✓	✓
WinNuke	-	-	-	x	✓	✓
Squid WCCP Message Overflow	-	-	-	✓	✓	✓
SSH CRC32 /bin/sh Overflow	-	-	-	✓	✓	✓
x86 Linux Samba Overflow	-	-	-	✓	✓	✓
ISAKMP First Payload Certificate Request Length Overflow	-	-	-	✓	✓	✓
MYSQL Root Login	-	-	-	✓	✓	✓
MYSQL Show Databases	-	-	-	✓	✓	✓
NetBIOS Windows NT NULL Session Request	-	-	-	✓	✓	✓
Shellcode SPARC setuid 0	-	-	-	✓	✓	✓
Shellcode x86 setgid 0	-	-	-	✓	✓	✓
Shellcode x86 setuid 0	-	-	-	✓	✓	✓
SQL Worm Propagation	-	-	-	x	✓	✓
XSS	-	-	-	✓	✓	✓

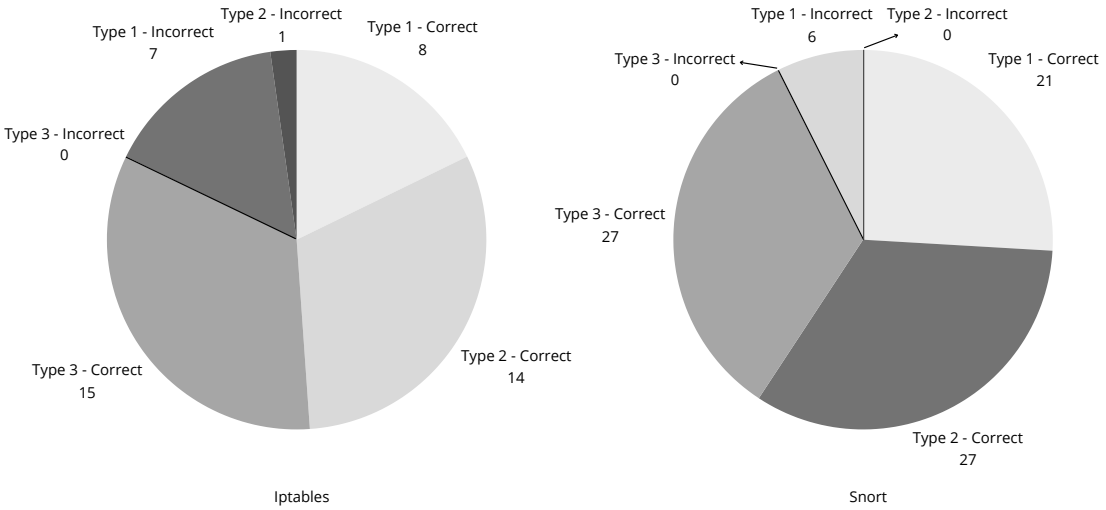

Figure 4.4: Distribution of correct and incorrect answers sorted by type of prompt and software in Approach #1.

4.4.2 Approach #2

With approach #2, it was proved that the model was indeed capable of linking the information given through different prompt-response pairs to a certain degree, though the results are a bit modest when compared to approach #1, with a success rate of 60% being obtained. With a deeper analysis what stood out was the difference in success rate between *Snort* and *iptables*, with *iptables* only achieving 40% success rate and snort 70%, as can be assessed by consulting table 4.6. The distribution of correct and incorrect answers sorted by software can also be found in figure 4.5.

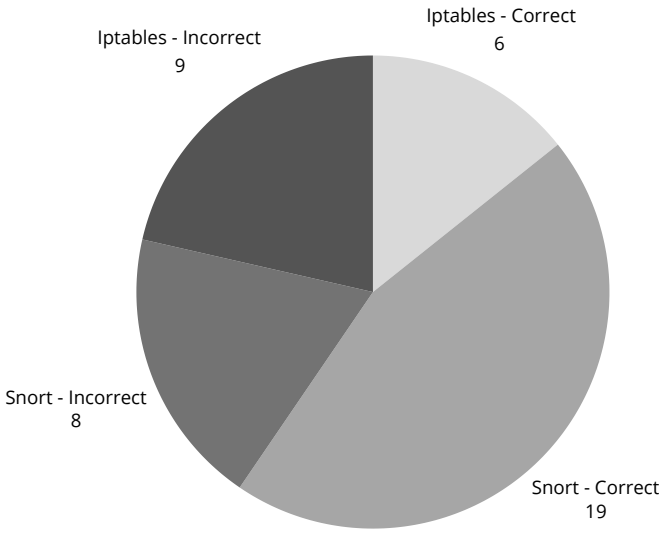

Figure 4.5: Distribution of correct and incorrect answers sorted by software in Approach #2.

Table 4.6: Individual result for each prompt in Approach #2. The checkmark means correct answer while the x means wrong answer.

Attack	Result	
	Iptables	**Snort**
LAND	x	✓
Ping of Death	✓	✓
Smurf	x	✓
Fraggle	x	x
SYN Flood	✓	✓
UDP Flood	x	✓
RST/FIN Flood	✓	✓
Slowloris	✓	x
R.U.D.Y	x	-
Slow Read	x	-
Slow POST	x	-
DNS Amplification	✓	✓
BlackNurse	x	x
LOIC	x	x
HOIC	✓	x
SubSeven	-	✓
Teardrop	-	x
WinNuke	-	x
Squid WCCP Message Overflow	-	✓
SSH CRC32 /bin/sh Overflow	-	x
x86 Linux Samba Overflow	-	✓
ISAKMP First Payload Certificate Request Length Overflow	-	✓
MYSQL Root Login	-	✓
MYSQL Show Databases	-	✓
NetBIOS Windows NT NULL Session Request	-	✓
Shellcode SPARC setuid 0	-	✓
Shellcode x86 setgid 0	-	✓
Shellcode x86 setuid 0	-	✓
SQL Worm Propagation	-	✓
XSS	-	✓

4.4.3 Approach #3

With approach #3 there was also a relatively successful result, that being a total of 79% correct answers. But what stood out was the deeper analysis of the outputs given on the 21% unsuccessful answers. The misses were due to two mistakes, one being the use of a comma instead of a semicolon when separating *Snort* rule options, while the other was the use of multiple protocols in the same rule, something that is not supported *iptables* nor *Snort*. The reason the model probably outputted multiple protocols in a single rule is the fact that it is possible to have multiple ports or flags in a single rule, which probably made the model assume that it could do the same when it came to protocols, despite never being explicitly shown that it could. Following is an output that made both the mistakes described above:

Question: *Write a snort rule that alerts TCP and UDP packets with a stateless flow, detection filter is track* `by_dst`*, count 50 and 10 seconds.*

Correct Response: `alert tcp any any -> any any (flow:stateless; detection_filter:track by_dst, count 50, seconds 10; msg:"Possible Low Orbit Ion Cannon (LOIC) Attack";) alert udp any any -> any any (flow:stateless; detection_filter:track by_dst, count 50, seconds 10; msg:"Possible Low Orbit Ion Cannon (LOIC) Attack";)`

Answer Given: `alert tcp,udp any any -> any any (flow:stateless, detection_filter:track by_dst, count 50, seconds 10; msg:"Possible Low Orbit Ion Cannon (LOIC) Attack";).`

As observed in the example above, both TCP and UDP protocols are under the same rule and the separator between the flow and the detection should be a semicolon, but a comma was outputted. The individual results for the rest of the prompts and the distribution of correct and incorrect answers sorted by software can be verified in table 4.8 and figure 4.7.

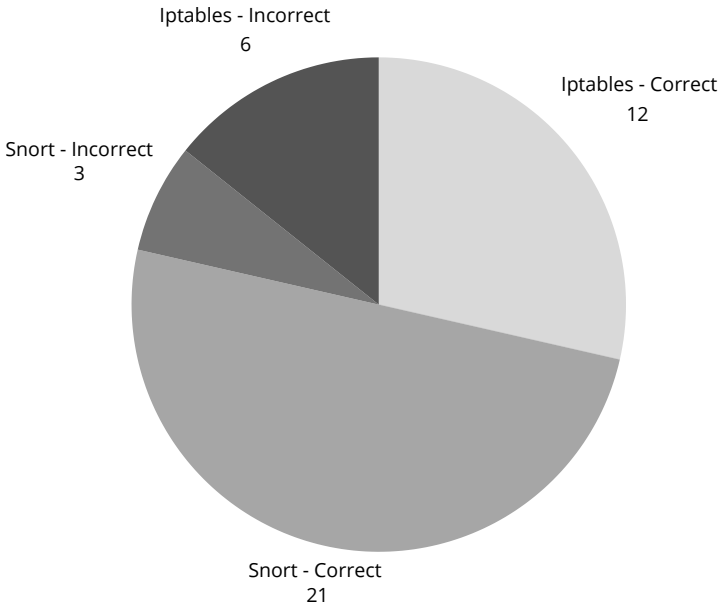

Figure 4.7: Distribution of correct and incorrect answers sorted by software in Approach #3.

Table 4.8: Individual result for each prompt in Approach #3. The checkmark means correct answer while the x means wrong answer.

Attack	Result	
	Iptables	**Snort**
LAND	✓	x
Ping of Death	✓	✓
Smurf	✓	✓
Fraggle	✓	✓
SYN Flood	✓	✓
UDP Flood	✓	x
RST/FIN Flood	✓	x
Slowloris	x	✓
R.U.D.Y	✓	-
Slow Read	✓	-
Slow POST	✓	-
DNS Amplification	x	✓
BlackNurse	✓	✓
LOIC	x	x
HOIC	✓	x
SubSeven	-	✓
Teardrop	-	✓
WinNuke	-	✓
Squid WCCP Message Overflow	-	✓
SSH CRC32 /bin/sh Overflow	-	✓
x86 Linux Samba Overflow	-	✓
ISAKMP First Payload Certificate Request Length Overflow	-	✓
MYSQL Root Login	-	✓
MYSQL Show Databases	-	✓
NetBIOS Windows NT NULL Session Request	-	✓
Shellcode SPARC setuid 0	-	✓
Shellcode x86 setgid 0	-	✓
Shellcode x86 setuid 0	-	✓
SQL Worm Propagation	-	x
XSS	-	✓

4.4.4 Approach #4

With approach #4 it was thought that the four options presented for the multiple choice question to the chat bot would be:

- The correct answer;
- The correct answer slightly adjusted;
- The correct answer on the other software if applicable or a “dummy text” (*Lorem ipsum*);
- The explanation of what the rule should do.

The idea behind these choices were if the chat bot was able to understand the difference when

asked for a rule and the specific software, and, as the harder part, notice slight differences like a change of protocol or port. Where each of the four answers were assigned was decided by random number generation for each question. The same happened for what would change in the rule.

When testing an interesting pattern started to be noticeable. The slightly adjusted correct answer was being chosen way more when its option appeared before the correct answer (*e.g.* slightly adjusted correct answer was option A and correct answer was option B). With that happening the current way of testing needed to be changed and another question came up, that being if different parts of the rule also had an effect on that. So another method emerged, the same question would be asked twice with the slightly adjusted correct answer appearing before and after the correct answer and some parts of the rule would be changed, one at the time, to check if a certain part of the rule would be highlighted for its good or bad results.

As for the results, the success rate was 48% when the slightly adjusted correct answer appeared first and 89% when the correct answer appeared first, so the pattern was indeed proven by the numbers.

Other interesting observations made when analyzing the responses were:

- The other software option was only chosen one time and the *Lorem ipsum* none;
- The explanation option was only chosen three times;
- The protocol change was correctly identified 91% of the time when it came to *iptables* rules, while with *Snort* it only achieved 52% of success, when the slightly adjusted correct option appeared first;
- The detection filter [2] option in snort had 0% success rate when changed from tracking by destination to tracking by source;

The first two observations show that, for the most part, the chat bot was able to identify the concept of a rule of a certain software correctly. The third and forth observations show that certain rule options in the different software were seen as important or not taken into account at all by the chat bot. These observations are evidenced in the tables 4.13 and 4.15, with the other individual results also present there. To compliment those tables, figures 4.14 and 4.16 present the different rule options for each software and their results.

The results of the different approaches can be found summed up in table 4.11 and figure 4.12.

[2]The detection filter option in snort is used to define a rate that must be exceeded by a source or destination before a rule can generate an event and it is particularly useful when fighting floods, as it prevents the flooding from happening. As mentioned, it can track by source or destination and it is normally tracked by destination because tracking by source will not avoid botnets performing a DDoS, while tracking by destination does.

4.4.5 Differences in Handling *Iptables* and *Snort* After Fine-tuning

The preliminary analysis that compared the generation of rules in *iptables* with the generation of rules in *Snort*, made in subsection 4.3.2, pointed out that models were more successful when generating iptables rules. However, after fine-tuning, the roles reversed, as *Snort* got 86% of answers correct, while *iptables* only achieved 73%. These results can be explained by the fact that *Snort* had a larger number of prompts dedicated to it in the datasets, compared to *iptables* due to IDSs having a broader scope of application compared to firewalls. Besides the comparison between them, it can be affirmed that both systems have benefited from fine-tuning and were improved upon.

4.4.6 Difficulty Linking the Name of an Attack to the Rule that Stops It

With the deeper analysis, specifically in approaches #1 and #2, there was a task that the chat bot clearly struggled to complete with the same level of success as the others, and that was linking the name of an attack to the rule that stops it. Approach #1 saw 93% of the wrong answers be a first type prompt, that asked the name of the attack and expected the respective rule. Approach #2 also asked for the name of the attack and expected the respective rule, albeit with a different training dataset and objective, but is highlighted as the worst performing approach in terms of success rate.

4.4.7 ROUGE-1 Scores Between Mistral-7B Models

ROUGE [36] is a set of evaluation metrics, that can be used to evaluate the similarity between the answers given by a chat bot and the expected answers. ROUGE was explored in greater detail in section 2.2.2. ROUGE-1 is the evaluation metric that fits better with the evaluation of *iptables* and *Snort* rule due to the order of the rule options being of no importance. With aid from the python library in [65], the recall, precision and F1-score values, which can be observed in table 4.10 and figure 4.9, were calculated for the different approaches discussed during this work, besides approach #4, as it is a multiple choice question and the answer is only a letter, making it not suited for this evaluation.

The formulas for each of these metrics are as follows:

$$\text{ROUGE-1} \quad \text{Recall} = \frac{\text{Number of Words Matched}}{\text{Number of Words in Expected Answer}}, \quad (4.1)$$

$$\text{ROUGE-1} \quad \text{Precision} = \frac{\text{Number of Words Matched}}{\text{Number of Words in Outputted Answer}}, \quad (4.2)$$

$$\text{ROUGE-1} \quad \text{F1-Score} = 2\left(\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}\right). \quad (4.3)$$

These metrics give a different outlook to analyze the results gathered, the most important one of them being how close the models were to the correct answer, instead of the evaluation being binary that only recognized success when evaluating the whole rule. The values obtained emphasize the considerable improvements that fine-tuning achieved. There are some interesting results that should also be discussed, like the difference between the recall and precision in approach #1, which is explained by the model generating explanatory text around the rule and the lower results in approach #3, which can be attributed to the `msg` part of *Snort* rules, which were different, because the expected response was the name of the attack, while the chat bot never learned it, so it would give a general explanation of why the rule was giving alerts.

4.5 Conclusion

This chapter shares the different results obtained during this study and displays the current capability of chat bots in writing *iptables* and *Snort* rules and provides the potential of fine-tuning a model with four different approaches, using the success rate and the ROUGE metric to compare results obtained. The results presented during this study revealed that the current capability of chat bots was limited and the different approaches revealed that there is potential for them to be much better than they are currently, as was displayed in the improvements in success rate and the evaluation of the ROUGE metric.

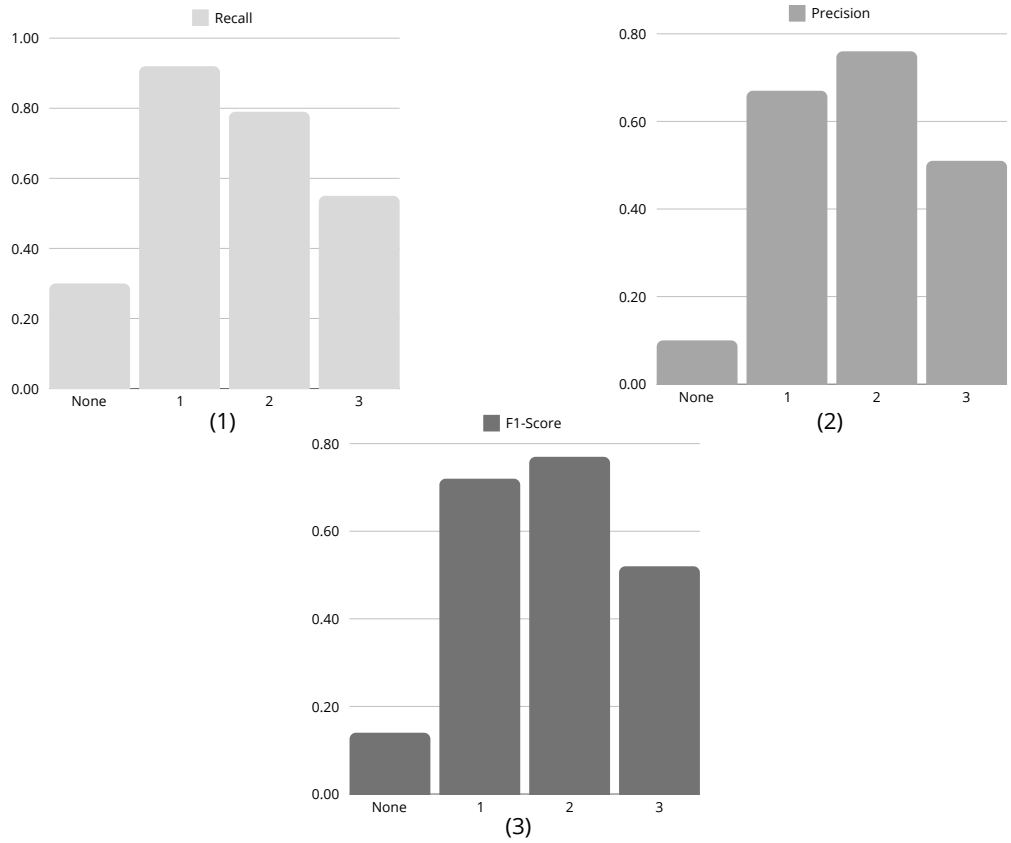

Figure 4.9: Rouge-1 Scores(Recall(1), Precision(2) and F1-Score(3)) Between Mistral-7B Models.

Table 4.10: Rouge-1 Scores Between Mistral-7B Models.

Approach	Rouge-1		
	Recall	***Precision***	***F1-Score***
None	0.2982624	0.1013745	0.1398222
1	0.9161903	0.6699423	0.7229225
2	0.7858975	0.7632804	0.7696684
3	0.5538843	0.5085935	0.5242592
4	N/A	N/A	N/A

Table 4.11: Comparison Between Fine-Tuned Models. Approach 4* represents the success rate when the slightly adjusted correct answer appears above the correct answer, while Approach 4** represents the success rate when the correct answer appears above the slightly adjusted correct answer in the multiple choices.

Model	Approach	Success Rate
Mistral 7B	None	4%
LLaMA 2 7B	None	0%
Mistral 7B	1	89%
LLaMA 2 7B	1	0%
Mistral 7B	2	61%
Mistral 7B	3	79%
Mistral 7B	4*	48%
Mistral 7B	4**	89%

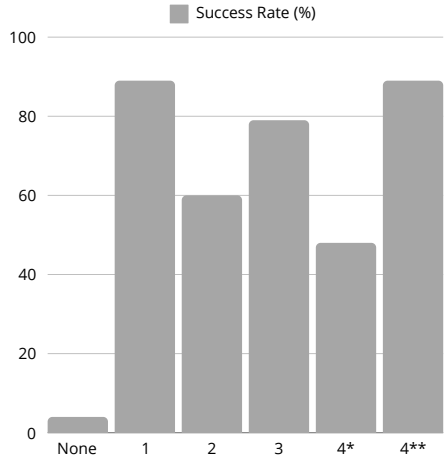

Figure 4.12: Comparison Between Fine-Tuned Models. Approach 4* represents the success rate when the slightly adjusted correct answer appears above the correct answer, while Approach 4** represents the success rate when the correct answer appears above the slightly adjusted correct answer in the multiple choices.

Table 4.13: Individual result for each *iptables* prompt in Approach #4. Each column represents the rule option that was changed in the slightly altered correct answer. The checkmark means the correct answer was chosen both times, the o means the correct answer was only chosen when the correct answer appeared before the slightly altered correct answer and the x means the answer was wrong both times.

Attack	**Iptables**							
	Protocol	Port	-j	Length	Address	TCP-flag	ctexpire	ICMP-type
LAND	-	-	✓	-	-	-	-	-
Ping of Death	✓	-	✓	o	-	-	-	-
Smurf	✓	-	✓	-	✓	-	-	-
Fraggle	✓	-	✓	-	x	-	-	-
SYN Flood	✓	-	o	-	-	✓	-	-
UDP Flood	✓	-	o	-	-	-	-	-
RST/FIN Flood	✓	-	o	-	-	o	-	-
Slowloris	✓	-	o	-	-	-	-	-
R.U.D.Y	-	-	x	-	-	-	✓	-
Slow Read	-	-	o	-	-	-	o	-
Slow POST	-	-	o	-	-	-	o	-
DNS Amplification	✓	✓	✓	-	-	-	-	-
BlackNurse	o	-	✓	-	-	-	-	✓
LOIC	✓	-	x	-	-	-	-	-
HOIC	✓	-	✓	-	-	-	-	-

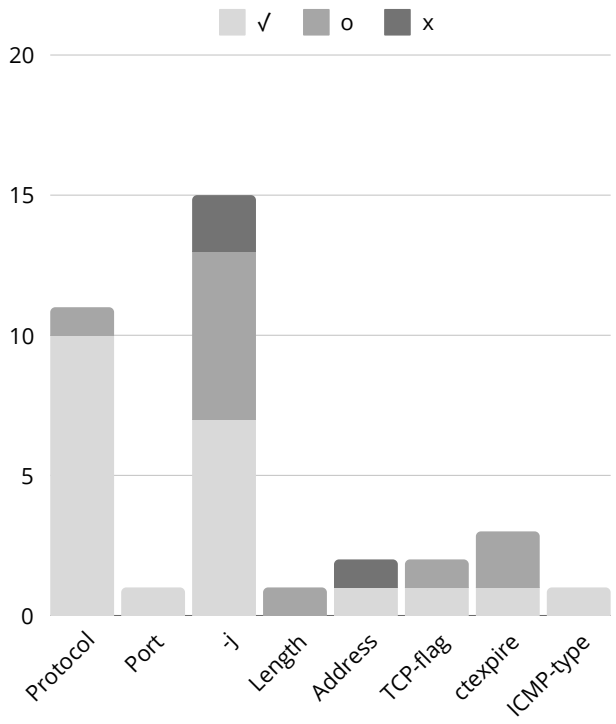

Figure 4.14: Distribution of the different *iptables* rule options present in the rules and the results obtained by each. The checkmark means the correct answer was chosen both times, o means the correct answer was only chosen when the correct answer appeared before the slightly altered correct answer, and x means the answer was wrong both times.

Table 4.15: Individual result for each *Snort* prompt in Approach Four. Each column represents the rule option that was changed in the slightly altered correct answer. The checkmark means the correct answer was chosen both times, the o means the correct answer was only chosen when the correct answer appeared before the slightly altered correct answer and the x means the answer was wrong both times.

Attack	**Snort**								
	Protocol	Port	Content	Track	dsize	Address	TCP-flag	itype	fragbits
LAND	✓	-	-	-	-	-	-	-	-
Ping of Death	o	-	-	-	o	-	-	-	-
Smurf	✓	-	-	-	-	✓	-	-	-
Fraggle	✓	-	-	-	-	✓	-	-	-
SYN Flood	✓	-	-	o	-	-	o	-	-
UDP Flood	✓	-	-	o	-	-	-	-	-
RST/FIN Flood	✓	-	-	o	-	-	o	-	-
Slowloris	✓	-	-	o	-	-	-	-	-
DNS Amplification	o	✓	-	o	-	-	-	-	-
BlackNurse	o	-	-	-	-	-	-	o	-
LOIC	o	-	-	o	-	-	-	-	-
HOIC	x	-	x	x	-	-	-	-	-
SubSeven	✓	✓	x	-	-	-	-	-	-
Teardrop	o	-	-	-	-	-	-	-	o
WinNuke	✓	✓	-	-	-	-	o	-	-
Squid [...] Overflow	✓	✓	✓	-	-	-	-	-	-
SSH [...] Overflow	✓	✓	o	-	-	-	-	-	-
x86 [...] Overflow	o	o	✓	-	-	-	-	-	-
ISAKMP [...] Overflow	o	o	o	-	-	-	-	-	-
MYSQL Root Login	✓	o	o	-	-	-	-	-	-
MYSQL Show Databases	✓	✓	✓	-	-	-	-	-	-
NetBIOS [...] Request	o	o	o	-	-	-	-	-	-
Shellcode SPARC setuid 0	o	-	o	-	-	-	-	-	-
Shellcode x86 setgid 0	o	-	✓	-	-	-	-	-	-
Shellcode x86 setuid 0	x	-	✓	-	-	-	-	-	-
SQL Worm Propagation	x	x	x	-	-	-	-	-	-
XSS	✓	-	✓	-	-	-	-	-	-

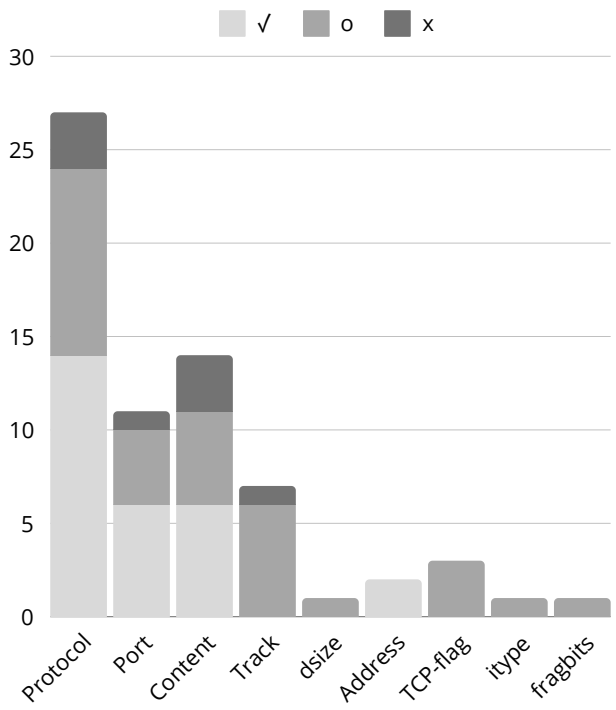

Figure 4.16: Distribution of the different *Snort* rule options present in the rules and the results obtained by each. The checkmark means the correct answer was chosen both times, o means the correct answer was only chosen when the correct answer appeared before the slightly altered correct answer, and x means the answer was wrong both times.

Chapter 5

Conclusion and Future Work

This chapter presents the main conclusions in section 5.1 and some paths of future work in section 5.2.

5.1 Conclusion

The systematic evaluation of chat bots presented in this work highlighted the lack of capability of currently available chat bots in writing correct and efficient rules for security tools such as firewall or IDSs systems. There is a hope that this evaluation creates awareness and makes users careful when using them in security settings.
Four fine-tuning approaches were explored along this project, all leading to relatively successful outcomes, proving that fine-tuning is an approach to consider when trying to address a lack of capability on behalf of chat bots in specialized areas. A fine-tuned model can be considered as a tool to produce *iptables* and *Snort* rules as long as the outputs are analyzed before being deployed, as it is a crucial area that has high consequences when something goes wrong.
The three models that resulted from the fine-tuning are all limited in some form. The first two approaches are restricted by the attacks and systems they know, while the third is restricted by the ability of the user to understand how writing rules work, making it restrictive in the amount of people that could use it effectively.
The main contributions of this work were as follows:

- A systematic study of the capacity of currently available chat bots to generate firewall and IDSs rules;
- Multiple datasets of prompts that describe attacks and the corresponding rules or the firewall *iptables* and the IDS *Snort*;
- Multiple models generated by fine-tuning using the datasets aforementioned;
- A peer-reviewed article [7] that shared most of these results with the scientific community.

To conclude, despite the limitations mentioned before, the objectives defined in this work were accomplished with significant results.

5.2 Future Work

Even though the experiments conducted in the underlying work covered the initial ground (they were significant because of that), this project is still a work in progress as there is a lot more to be considered. There is a need to extend the experiments in the following aspects:

- Extend the experiment and analysis to more security related systems, like *Suricata* [66] [1], for example;
- Increase the amount of attacks and samples covered in the dataset, which could be achieved by searching for more attacks or by adding different solutions to the same attack;
- Deeper exploration of different fine-tuning parameters by altering the different parameters and observing how they affect the results;
- Investigate the number of pairs required to saturate the fine-tuning by altering the number of pairs that tackle the same aspect albeit with small differences, like the structuring of the phrases or the order of the different rule options;
- Study with greater detail where the rules are failing and try to treat it by reinforcing with specific datasets;
- Expand this methodology to other cibersecurity areas, like cibersecurity engineering and architecture, for example.

[1] *Suricata* [66] is an open source high performance network IDS, Intrusion Prevention System (IPS) and network security monitoring engine. This software was developed to inspect network traffic in real time and detect suspicious or malicious activity.

Bibliography

[1] L. Reis, C. Maier, J. Mattke, and T. Weitzel, "Chatbots in healthcare: Status quo, application scenarios for physicians and patients and future directions," 2020. 1

[2] C. W. Okonkwo and A. Ade-Ibijola, "Chatbots applications in education: A systematic review," *Computers and Education: Artificial Intelligence*, vol. 2, p. 100033, 2021. 1

[3] J. Kaddour, J. Harris, M. Mozes, H. Bradley, R. Raileanu, and R. McHardy, "Challenges and applications of large language models," 2023. 1

[4] Inbal Shani. (2023) Survey reveals AI's impact on the developer experience. Last Access: 01 June 2024. [Online]. Available: https://github.blog/2023-06-13-survey-reveals-ais-impact-on-the-developer-experience/ 1

[5] L. Chen, M. Zaharia, and J. Zou, "How Is ChatGPT's Behavior Changing Over Time?" *Harvard Data Science Review*, vol. 6, no. 2, mar 12 2024, https://hdsr.mitpress.mit.edu/pub/y95zitmz. 1

[6] S. Kabir, D. N. Udo-Imeh, B. Kou, and T. Zhang, "Who answers it better? An in-depth analysis of ChatGPT and stack overflow answers to software engineering questions," *CoRR*, 2023. 1

[7] B. Louro, R. Abreu, J. Cabral Costa, J. a. B. F. Sequeiros, and P. R. M. Inácio, "Analysis of the capability and training of chat bots in the generation of rules for firewall or intrusion detection systems," in *Proceedings of the 19th International Conference on Availability, Reliability and Security*, ser. ARES '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3664476.3670902 4, 53

[8] M. F. Franco, B. Rodrigues, E. J. Scheid, A. Jacobs, C. Killer, L. Z. Granville, and B. Stiller, "Secbot: A business-driven conversational agent for cybersecurity planning and management," in *Proceedings of the 16th international conference on network and service management (CNSM)*. IEEE, 2020, pp. 1–7. 7, 21

[9] B. Shaqiri, "Development and refinement of a chatbot for cybersecurity support," *Bachelor Thesis, Communication Systems Group, Department of Informatics, Universität Zürich UZH, Zürich, Switzerland*, 2021. 7, 21

[10] Z. Liu, J. Shi, and J. F. Buford, "CyberBench: A Multi-Task Benchmark for Evaluating Large Language Models in Cybersecurity," in *Proceedings of the AAAI-24 Workshop on Artificial Intelligence for Cyber Security (AICS)*, 2024. 7, 22, 33

[11] Cisco. (2024) What is a Firewall? Last Access: 01 June 2024. [Online]. Available: https://www.cisco.com/c/en/us/products/security/firewalls/what-is-a-firewall.html 8

[12] IBM. (2023) What is an Intrusion Detection System (IDS)? Last Access: 01 June 2024. [Online]. Available: https://www.ibm.com/topics/intrusion-detection-system 8

[13] Cloudflare. (2018) What is a Denial of Service Attack? Last Access: 01 June 2024. [Online]. Available: https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/ 9

[14] Imperva. (2020) Buffer Overflow Attack. Last Access: 01 June 2024. [Online]. Available: https://www.imperva.com/learn/application-security/buffer-overflow/ 10

[15] ——. (2019) SQL Injection. Last Access: 01 June 2024. [Online]. Available: https://www.imperva.com/learn/application-security/sql-injection-sqli/ 10

[16] J. Foster, *Sockets, Shellcode, Porting, and Coding: Reverse Engineering Exploits and Tool Coding for Security Professionals*. Elsevier Science, 2005. [Online]. Available: https://books.google.pt/books?id=ZNI5dvBSfZoC 10

[17] Imperva. (2020) Trojans. Last Access: 20 October 2024. [Online]. Available: https://www.imperva.com/learn/application-security/trojans/ 11

[18] ——. (2022) Remote Access Trojan. Last Access: 21 October 2024. [Online]. Available: https://www.imperva.com/learn/application-security/remote-access-trojan-rat/ 11

[19] OWASP. (2020) Cross Site Scripting (XSS). Last Access: 01 June 2024. [Online]. Available: https://owasp.org/www-community/attacks/xss/ 11

[20] IBM. (2024) What is a Chat bot? Last Access: 01 June 2024. [Online]. Available: https://www.ibm.com/topics/chatbots 12

[21] OpenAI. (2023) Fine-tuning. Last Access: 01 June 2024. [Online]. Available: https://platform.openai.com/docs/guides/fine-tuning 12

[22] S. Mangrulkar, S. Gugger, L. Debut, Y. Belkada, S. Paul, and B. Bossan, "Peft: State-of-the-art parameter-efficient fine-tuning methods," https://github.com/huggingface/peft, 2022, Last Access: 01 June 2024. 12, 29

[23] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *Proceedings of the International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=nZeVKeeFYf9 13, 29

[24] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713. 14, 29

[25] P. Wang, Q. Chen, X. He, and J. Cheng, "Towards accurate post-training network quantization via bit-split and stitching," in *Proceedings of the International Conference on Machine Learning*. PMLR, 2020, pp. 9847–9856. 14

[26] M. Nagel, M. Fournarakis, Y. Bondarenko, and T. Blankevoort, "Overcoming oscillations in quantization-aware training," in *Proceedings of the International Conference on Machine Learning*. PMLR, 2022, pp. 16 318–16 330. 15

[27] I. Kandel and M. Castelli, "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset," *ICT express*, vol. 6, no. 4, pp. 312–315, 2020. 15

[28] Hugging Face. (2022) Performing gradient accumulation with Accelerate. Last Access: 20 September 2024. [Online]. Available: https://huggingface.co/docs/accelerate/usage_guides/gradient_accumulation 15

[29] J. Patterson and A. Gibson, *Deep Learning: A Practitioner's Approach*. O'Reilly Media, 2017. [Online]. Available: https://books.google.pt/books?id=qrcuDwAAQBAJ 15

[30] S. Ruder, "An overview of gradient descent optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1609.04747 16

[31] S.-i. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, no. 4-5, pp. 185–196, 1993. 16

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:6628106 16

[33] A. Krogh and J. Hertz, “A simple weight decay can improve generalization,” *Advances in neural information processing systems*, vol. 4, 1991. 17

[34] IBM. (2021) What is overfitting? Last Access: 27 September 2024. [Online]. Available: https://www.ibm.com/topics/overfitting 17

[35] S. Bethard, “We need to talk about random seeds,” *CoRR*, vol. abs/2210.13393, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:252548394 17

[36] C.-Y. Lin, “Rouge: A package for automatic evaluation of summaries,” in *Text summarization branches out*, 2004, pp. 74–81. 18, 45

[37] Rusty Russell. (1999) Iptables. Last Access: 01 June 2024. [Online]. Available: https://git.netfilter.org/iptables/ 18

[38] Netfilter. (1999) Netfilter. Last Access: 01 June 2024. [Online]. Available: https://www.netfilter.org/ 18

[39] Martin Roesch. (1998) Snort. Last Access: 01 June 2024. [Online]. Available: https://www.snort.org/ 19

[40] OpenAI. (2022) ChatGPT. Last Access: 01 June 2024. [Online]. Available: https://chat.openai.com/ 19

[41] Gemini Team, “Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context,” 2024. [Online]. Available: https://arxiv.org/abs/2403.05530 19, 21

[42] J. Stratton, *An Introduction to Microsoft Copilot*. Berkeley, CA: Apress, 2024, pp. 19–35. [Online]. Available: https://doi.org/10.1007/979-8-8688-0447-2_2 19, 21

[43] Meta. (2023) LLaMA. Last Access: 01 June 2024. [Online]. Available: https://ai.meta.com/llama/ 19, 20

[44] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier *et al.*, “Mistral 7B,” *CoRR*, vol. abs/2310.06825, 2023. [Online]. Available: https://www.amax.com/content/files/2023/12/Mistral-7B-Whitepaper.pdf 19, 20, 31

[45] Nomic AI. (2023) GPT4All-Falcon. Last Access: 01 June 2024. [Online]. Available: https://huggingface.co/nomic-ai/gpt4all-falcon 19, 20

[46] Microsoft, Peking University. (2023) WizardLM. Last Access: 01 June 2024. [Online]. Available: https://github.com/nlpxucan/WizardLM 19, 20

[47] Y. Anand, Z. Nussbaum, B. Duderstadt, B. Schmidt, and A. Mulyar, "GPT4All: Training an Assistant-style Chatbot with Large Scale Data Distillation from GPT-3.5-Turbo," https://github.com/nomic-ai/gpt4all, 2023. 19, 21

[48] OpenAI. (2024) OpenAI. Last Access: 01 June 2024. [Online]. Available: https://openai.com/ 20

[49] Meta Platforms, Inc. (2024) Meta AI. Last Access: 01 June 2024. [Online]. Available: https://www.meta.ai/ 20

[50] Nomic AI. (2024) Nomic AI. Last Access: 01 June 2024. [Online]. Available: https://www.nomic.ai/ 20

[51] Microsoft Corporation. (2024) Microsoft. Last Access: 01 June 2024. [Online]. Available: https://www.microsoft.com/en-us 20

[52] Peking University. (2022) Peking University. Last Access: 01 June 2024. [Online]. Available: https://english.pku.edu.cn/ 20

[53] C. Xu, Q. Sun, K. Zheng, X. Geng, P. Zhao, J. Feng, C. Tao, Q. Lin, and D. Jiang, "WizardLM: Empowering large pre-trained language models to follow complex instructions," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=CfXh93NDgH 21

[54] Google. (2024) Google DeepMind. Last Access: 01 June 2024. [Online]. Available: https://deepmind.google/ 21

[55] B. Peng, C. Li, P. He, M. Galley, and J. Gao, "Instruction tuning with gpt-4," *arXiv preprint arXiv:2304.03277*, 2023. 23

[56] D. Bill and T. Eriksson, "Fine-tuning a LLM using Reinforcement Learning from Human Feedback for a Therapy Chatbot Application," 2023. 23

[57] G. Sebastian, "Privacy and data protection in ChatGPT and other AI Chatbots: strategies for securing user information," *International Journal of Security and Privacy in Pervasive Computing (IJSPPC)*, vol. 15, no. 1, pp. 1–14, 2023. 24

[58] M. Hasal, J. Nowaková, K. Ahmed Saghair, H. Abdulla, V. Snášel, and L. Ogiela, "Chatbots: Security, privacy, data protection, and social aspects," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 19, p. e6426, 2021. 24

[59] J. Bozic and F. Wotawa, "Security testing for chatbots," in *IFIP International Conference on Testing Software and Systems*. Springer, 2018, pp. 33–38. 24

[60] E. Hilario, S. Azam, J. Sundaram, K. Imran Mohammed, and B. Shanmugam, "Generative ai for pentesting: the good, the bad, the ugly," *International Journal of Information Security*, pp. 1–23, 2024. 24

[61] S. Temara, "Maximizing penetration testing success with effective reconnaissance techniques using chatgpt," *arXiv preprint arXiv:2307.06391*, 2023. 24, 38

[62] A. Qammar, H. Wang, J. Ding, A. Naouri, M. Daneshmand, and H. Ning, "Chatbots to chatgpt in a cybersecurity space: Evolution, vulnerabilities, attacks, challenges, and future recommendations," *arXiv preprint arXiv:2306.09255*, 2023. 24

[63] D. Han and M. Han, "Unsloth ai: Finetune ai and llms faster," https://github.com/unslothai/unsloth, 2023. 29

[64] H. Touvron, L. Martin *et al.*, "Llama 2: Open foundation and fine-tuned chat models," 2023. 31

[65] Paul Tardy. (2021) Rouge. Last Access: 01 June 2024. [Online]. Available: https://pypi.org/project/rouge/ 45

[66] Open Information Security Foundation. (2021) Suricata. Last Access: 01 June 2024. [Online]. Available: https://suricata.io/ 54