UNIVERSITY OF THE PELOPONNESE & NCSR “DEMOCRITOS”
MSC PROGRAMME IN DATA SCIENCE

ChatBot for Cyber-Security based on RASA Framework

by

Alexandros Marantos

A thesis submitted in partial fulfillment
of the requirements for the MSc
in Data Science

Supervisor: Christos Tryfonopoulos
Professor

Co-supervisors: Georgios Petasis, Researcher C
Nikolaos Platis, Assistant Professor

Athens, June 2024

ChatBot for Cyber-Security based on RASA Framework

Alexandros Marantos

MSc. Thesis, MSc. Programme in Data Science

University of the Peloponnese & NCSR “Democritos”, June 2024

UNIVERSITY OF THE PELOPONNESE & NCSR “DEMOCRITOS”
MSC PROGRAMME IN DATA SCIENCE

ChatBot for Cyber-Security based on RASA Framework

by

Alexandros Marantos

A thesis submitted in partial fulfillment
of the requirements for the MSc
in Data Science

Supervisor: Christos Tryfonopoulos
Professor

Co-supervisors: Georgios Petasis, Researcher C
Nikolaos Platis, Assistant Professor

Approved by the examination committee on June, 2024.

(Signature)	(Signature)	(Signature)
....	....	....
Christos Tryfonopoulos Professor	Georgios Petasis Researcher C	Nikolaos Platis Assistant Professor

Athens, June 2024

UNIVERSITY OF THE PELOPONNESE & NCSR "DEMOCRITOS"
MSC PROGRAMME IN DATA SCIENCE

Declaration of Authorship

(1) I declare that this thesis has been composed solely by myself and that it has not been submitted, in whole or in part, in any previous application for a degree. Except where states otherwise by reference or acknowledgment, the work presented is entirely my own.

(2) I confirm that this thesis presented for the degree of Bachelor of Science in Informatics and Telecommunications, has

(i) been composed entirely by myself

(ii) been solely the result of my own work

(iii) not been submitted for any other degree or professional qualification

(3) I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or processional qualification except as specified.

(Signature)

....

Alexandros Marantos

Athens, June 2024

Acknowledgments

I would like to thank my professor and thesis supervisor, Dr. Christos Tryfonopoulos, for his guidance, support and encouragement that kept me motivated throughout my research process. I would also like to thank my family, Ioanna, my friends and Giorgos for their support throughout my academic journey. Their emotional support has been a constant resource of courage.

I usually solve problems by letting them devour me.

Franz Kafka

Περίληψη

Με την ολοένα αυξανόμενη πολυπλοκότητα των απειλών και των ευπαθειών της κυβερνοασφάλειας, η ανάγκη για αποδοτικές και αποτελεσματικές λύσεις διαχείρισης αυτών είναι αδήριτη. Το chatbot κυβερνοασφάλειας που προτείνεται στην παρούσα εργασία επιδιώκει να εισαγάγει έναν πιο φιλικό προς τον χρήστη τρόπο εύρεσης και αναζήτησης πληροφοριών σχετικά με πολλαπλά είδη απειλών. Η παρούσα διπλωματική εργασία διερευνά την ανάπτυξη και υλοποίηση ενός νέου και καινοτόμου chatbot που αξιοποιεί τεχνικές επεξεργασίας φυσικής γλώσσας (NLP) για την αναζήτηση ευπαθειών κυβερνοασφάλειας.

Το chatbot που παρουσιάζεται στη παρούσα εργασία προσφέρει ποικίλες λειτουργίες, που περιλαμβάνουν από απλούς διαλόγους έως την εις βάθος διερεύνηση των απειλών για την ασφάλεια στον κυβερνοχώρο. Οι χρήστες μπορούν να πραγματοποιήσουν ερωτήσεις σχετικά με τα επίπεδα απειλών, συμπεριλαμβανομένων των ευπαθειών χαμηλής απειλής και των απειλών 0-day. Οι χρήστες μπορούν να εξερευνήσουν τα δεδομένα MISP, να λάβουν πληροφορίες για συγκεκριμένα γεγονότα ευπαθειών και συσχετίσεων αυτών αλλά και να εξερευνήσουν τις πηγές δεδομένων τους. Το chatbot παρέχει ολοκληρωμένες πληροφορίες σχετικά με συγκεκριμένες απειλές, όπως trojans, spywares, ιούς και adwares, συμπεριλαμβανομένης της λειτουργικότητας τους αλλά και πολλών άλλων πληροφοριών. Το chatbot διευκολύνει την ολοκληρωμένη ανάλυση απειλών, επιτρέποντας στους χρήστες να φιλτράρουν τις απειλές αυτές ημερολογιακά, με βάση τα επίπεδα κινδύνου και άλλα.

Τα ευρήματα της παρούσας εργασίας καταδεικνύουν τις δυνατότητες των chatbots ως πολύτιμα εργαλεία για τους επαγγελματίες της κυβερνοασφάλειας αλλά και τους απλούς χρήστες, προσφέροντας έναν φιλικό προς το χρήστη περιβάλλον και έναν αποτε-

λεσματικό τρόπο πρόσβασης και ανάλυσης των πληροφοριών σχετικά με τις ευπάθειες. Η παρούσα εργασία αναλύει τους περιορισμούς της υλοποίησης και περιγράφει πιθανές μελλοντικές κατευθύνσεις για την ενίσχυση των δυνατοτήτων της, όπως την επέκταση των πηγών δεδομένων του και την ενσωμάτωση πιο προηγμένων τεχνολογίων επεξεργασίας φυσικής γλώσσας. Επίσης, αυτή η μελέτη θα επιχειρήσει να συγκρίνει το παρόν chatbot με άλλα δημοφιλή γενικού σκοπού, όπως το ChatGPT της OpenAI και το Gemini της Google.

Abstract

With the ever-increasing complexity of cybersecurity threats and vulnerabilities, the need for efficient and effective vulnerability management solutions tends to be vital. This proposed cybersecurity chatbot seeks to introduce a more user-friendly way of getting insights about multiple kind of threats. This thesis explores the development and implementation of a novel chatbot leveraging Natural Language Processing (NLP) techniques for cybersecurity vulnerability searches.

The implemented chatbot offers diverse functionalities, ranging from regular generic dialogues to in-depth exploration of cybersecurity threats. Users can inquire about threat levels, including low-threat vulnerabilities and zero-day threats. Users can explore MISP data, retrieving specific events and their correlations. The chatbot provides comprehensive details about specific threats like trojans, spyware, viruses, and adware, including functionality and many more insights. The chatbot facilitates comprehensive threat analysis by enabling users to filter threats based on date ranges, risk levels, and severity.

The findings demonstrate the potential of chatbots as a valuable tool for cybersecurity professionals and regular users, offering a user-friendly and efficient way to access and analyze vulnerability information. This thesis discusses the limitations of the current implementation and outlines potential future directions for enhancing the chatbot's capabilities, expanding its data sources, and incorporating more advanced NLP techniques. Also this study will attempt to compare this domain-specific chatbot with other popular general-purposed ones like OpenAI's ChatGPT and Google's Gemini.

Contents

List of Tables

List of Figures

List of Abbreviations

NLP	Natural Language Processing
MISP	Malware Information Sharing Platform
URL	Uniform Resource Locator
GPT	Generative Pre-trained Transformer
BERT	Bidirectional Encoder Representations from Transformers
DIET	Dual Intent and Entity Transformer
LLM	Large Language Model
SQL	Structured Query Language
JSON	JavaScript Object Notation
IR	Intent Recognition
NLU	Natural Language Understanding

Chapter 1

Introduction

1.1 Chatbot definition

In the lexicon, a chatbot is defined as “A computer program designed to simulate conversation with human users, especially over the Internet” [1]. Similarly, other definitions describe chatbots as “A computer program designed to respond with conversational or informational replies to verbal or written messages from users.” [2].

The fundamental concept facilitated in every chatbot is its ability to engage with real human users, through text messages and to simulate a feeling of a real human to human conversation while providing thoughtful responses. As technology advanced the development of chatbots growed as well, aiming to simulate human-like interactions. While chatbots share a distant connection with early experiments like Eliza, which simulated a psychotherapist, their evolution has been marked by advancements in natural language processing and machine learning. Today, chatbots are used across different domains and provide personalized assistance and automated tasks such as customer support, appointment scheduling, and information retrieval.

In the last years, in addition to specialized chatbots designed for specific tasks, there are also general-purpose chatbots. These chatbots are designed to engage in conversations on a wide range of topics offering a more flexible approach to interaction. General-purposed chatbots are being trained in huge datasets, leverage extensive knowledge bases and sophisticated algorithms to understand and respond to a diverse array of user inquiries.

1.2 Evolution of Chatbots

Since the inception of the earliest computers, attemps have been made to devise a computer program capable of communicating, emulating and engaging in dialogue to closely resemble interaction with a real human. Alan Turing proposed a test in 1950 suggesting that a machine could be considered intelligent if it could exhibit behavior indistinguishable from that of a human during textual communication. This scientific question of Turing was considered as the inventive idea of chatbots [3].

The first chatbot introduced, was implemented from the Massachusetts Institute of Technology (MIT) in 1960, named ELIZA. ELIZA could simulate communication by responding to human provided sentences by using a pattern matching algorithm. Given that was the first attempt on chatbots, ELIZA had very limited conversational ability but was a crucial step to more advanced chatbots that would follow [4].

After ELIZA in the following years, more chatbots were created coming across as more advanced than their predecessor. PARRY, created in 1972, was a chatbot that could handle complicated assumptions but it was slow could not learn from conversations [5]. After those, in the late 1980s and the 1990s more chatbots were developed like Jabberwacky which was learning from rules and context. Another milestone in the development of chatbots came with Dr. Sbaitso in 1991 which was a text-to-speech engine implementing artificial intelligence speech synthesis [6] and A.L.I.C.E., mostly inspired from ELIZA, which relied on pattern matching based on Artificial Intelligence Markup Language (AIML).

After 2000 and especially at the beginning of the second decade chatbots and voicebots experienced a huge rise in their popularity. Crucial to that was that the major technology companies showed a huge interest in the field and created their own. Apple created on 2010 Siri as a Personal Assistant tool for its mobile operating system. Siri took voice commands and using artificial intelligence algorithms searched through the internet to answer the specific queries. Google was the second major company to create a voicebot personal assistant introducing Google Assistant on 2012, a better implementation of Siri, implemented inside Google's Android mobile operating system. Subsequently Amazon and Microsoft also created personal assistants named Alexa and Cortana respectively utilizing natural language processing (NLP) to recognize and answer voice inputs [7].

Another artificial intelligence chatbot that became famous is the short-lived Microsoft's Tay chatbot which was deployed on Twitter in 2016. Upon its launch Tay was trained by user tweets, which resulted in becoming a failure as it was providing racist and mysogynistic answers. The same day was terminated from Microsoft [8].

A huge leap forward was made in 2020 and has persisted to the present at the development of AI chatbots, from the introduction of Generative Pre-trained Transformer Three (GPT-3) from OpenAI. The same company introduced OpenAI Codex in 2021 and the famous ChatGPT in 2022. GPT-3's neural networks contain 175 billion machine learning parameters [9], while ChatGPT, which fine-tunes GPT-3, is using supervised and reinforcement methods [10]. Recently, OpenAI unveiled GPT-4 which is more powerful than ChatGPT and more accurate in analyzing and providing data. GPT-4 can also process input from images and provide answers on queries based on the images context [11].

The latest powerful chatbot was introduced by Google with Gemini, the successor of Bard. Gemini is Google AI's most powerful and versatile large language model (LLM), unveiled in December 2023. It surpasses its predecessor Bard in its ability to handle various modalities of information, including text, code, image, and audio [12].

1.3 Cybersecurity domain

Cybersecurity, in today's fast paced digital world, is a crucial defence mechanism against a growing number of threats and vulnerabilities both in personal and commercial worlds. As technology advancements accelerate and the vast amount of data produced become more valuable, the attempts of data theft grows. Furthermore, the growing popularity of smart devices connected to the internet (Internet of Things) create new ways of cyberattacks [13].

There are many different approaches and techniques for a cyberattacker to follow. *Malware* is a term encompassing malicious software and comes in many forms each one attempting to disrupt a systems functionality and gain control on it. *Spyware* gathers user data without consent often installed by Trojans which disguise themselves as legitimate programs. *Viruses* on the other hand unlike Trojans self-replicate by attaching to other files. Another important type of vulnerability is *Adware* which disrupts your experience with intrusive advertisements [14].

1.4 Motivation

In the last years powerful general-purpose artificial intelligence trained chatbots are introduced and gain vast popularity in just days [15]. As the public becomes more prominent on using these useful tools the industry implements more task specific chatbots to automate other procedures such as customer service.

The motivation behind the proposed cybersecurity chatbot arises from the huge amount of different cybersecurity databases which are currently available. While the existence of so many sources to check is of great significance, also presents a challenge for users. Exploring such amount of vast data is time consuming and can lead to useful data being overlooked. A chatbot designed to connect with these databases could significantly improve the user experience.

1.5 Thesis structure

The subsequent chapter will dig into a review of existing literature focusing on existing chatbots in the domain of cybersecurity. This review will attempt an analysis based on the chatbot's technological foundations and the expertise of the users targeted by their design.

Diving into the technical aspects, Chapter 3 will provide a detailed presentation of the databases integrated with the implemented chatbot and the data structures within these databases. Additionally, Chapter 4 will offer an exhaustive overview of the architectural design and the technologies used in the development of the chatbot.

Chapter 5 will provide a detailed overview of the possible dialogue capabilities supported from the chatbot. This analysis will provide an exhaustive exploration of all the information the chatbot can currently extract from the supported databases.

Finally, chapter 6 will conclude the discussion by highlighting modifications and innovative ideas that could lead to further enhance the chatbot's currently capabilities.

Chapter 2

Related Work

Conversational agents have been widely used in different domains and areas of interest. A survey providing an overview and comparison on various Natural Language Processing (NLP) techniques and outlining significant factors that impact the design of a chatbot has been presented by [16]. The authors emphasize the importance of tailoring chatbots to specific tasks. NLP techniques can change depending on the available data. Limiting the scope of the chatbots can avoid needing massive knowledge bases like those required for general-purpose agents. This targeted approach allows different fields to benefit from chatbots trained and designed for their unique needs.

2.1 Transformers

The introduction of transformer architecture has a significant impact on NLP field [17]. Transformers have overcome some limitations on existing NLP methods that are implemented using Convolutional Neural Networks [18], Recurrent Neural Networks (RNNs) [19] or even Long Short-Term Memory networks (LSTM) [20]. Unlike traditional models that process text sequentially, transformers can analyze inputs as entire sentences or paragraphs at once. This allows them to identify complex relationships between words even if they are far apart in the text [21]. This parallel processing characteristics make transformers well-suited for various natural language processing tasks including understanding emotional tones (sentiment analysis). These advancements have set transformer models as the major technology for powerful NLP tools like BERT (Bidirectional Encoder Representations from Transformers) [22] and GPT (Generative Pre-trained Transformer) [23]. Transformer architectures with their self-attention capabilities have undoubtedly pushed the boundaries of NLP. However it's important to mention that they represent just one approach within the ever-growing field of Large Language Models (LLMs).

2.2 Large Language Models

Large Language Models (LLMs) represent a breakthrough in Natural Language Processing (NLP). These models are trained on large text datasets, including immense volumes of language data allowing them to develop a sophisticated understanding of language context [24]. This fact allow LLMs to perform tasks requiring comprehension of context and generation of human-quality text that is both coherent and semantically meaningful [25].

Both GPT-style and BERT-style language models use the powerful transformer architecture to capture contextual dependencies and relationships within text. This architecture allows them to have great accuracy in many different NLP applications. However they are different in their training approaches. GPT models are trained to predict the next word in a sequence capturing long-range dependencies. On the other hand, BERT, utilizes a masked language modeling objective where it predicts hidden words based on surrounding context. This approach enables BERT to grasp the different meaning of words and phrases within a sentence.

Large language models (LLMs) have demonstrated remarkable success in various natural language processing (NLP) tasks including binary classification and Named Entity Recognition (NER). This, suggests their potential application within the Cybersecurity domain and in particular for tasks like text classification and entity recognition [26]. In Cybersecurity, text classification could mean categorizing text data based on its relevance to security threats while NER could focus on identifying specific entities within the text such as vulnerabilities and affected products.

2.3 Existing Chatbots

2.3.1 Chatbots based on Transformers

2.3.1.1 SecBot

SecBot is a conversational agent focused on cybersecurity planning and management proposed on 2020 [27]. SecBot is a chatbot built in the Rasa platform which is based on the two fundamental concepts of Intents and Entities. Intents specify what users want to access through the chat and Entities are used to extract specific terms or values from the given query. SecBot uses the Dual Intent and Entity Transformer (DIET) architecture [28] for intent classification and entity extraction as implemented by the Rasa framework [29].

2.3.1.2 Chatbot Sec

ChatBot Sec (CBS) aims to be a virtual information security advisor for users especially those with limited cybersecurity expertise [30]. It uses a JSON file as its knowledge database containing pre-written security advices structured in a tree-like format. Upon user's query, CBS first checks its cache for a similar past inquiry and response. If no match is found, CBS searches the knowledge base for keywords related to the user's question. Based on the search results CBS retrieves relevant advice. After that formats it into a user-friendly response and sends it back. Finally, CBS caches this interaction for future reference if a similar question arises.

2.3.2 Chatbots based on LLMs

Recent advancements in Large Language Models (LLMs) have revolutionized Natural Language Processing (NLP) due to their impressive ability to comprehend and generate human-like text. However, their performance can suffer in domains requiring specialized knowledge or access to external information. However, the attempt to detect chatbots utilising Large Language Models (LLMs) within the Cybersecurity domain have not yielded significant results. The following examples demonstrate the application of LLMs in chatbots across various fields.

2.3.2.1 LLM Chatbot for Cybersecurity

A novel chatbot leveraging Large Language Models (LLMs) to enhance anomaly detection in cybersecurity was proposed by **Balasubramanian et. al.** [31]. This chatbot combines advanced GPT-3 models together with rule-based logic to analyze system logs. By examining the aforementioned logs the chatbot can identify and extract unusual patterns or deviations that might lead to potential security threats and vulnerabilities. This approach combines machine learning capabilities with the expertise of cybersecurity professionals attempting to establish a new standard for anomaly-aware conversational agents. The evaluation of this chatbot demonstrates the system's effectiveness to have very high accuracy in identifying anomalies within logs. Furthermore GPT-3 models seem to consistently outperform other compared LLMs (BERT, DistilBERT, ALBERT) showcasing their superior performance in this domain.

2.3.2.2 LLM Chatbot for Blockchain

In the work presented by **Mansurova et.al.** [32] a novel system that addresses these limitations is discussed which integrates LLMs with an external knowledge management module. This module facilitates real-time access to external data sources for the LLM regarding the blockchain domain. More specifically, the system utilizes vector databases for efficient retrieval of relevant information while also enabling dynamic internet search capabilities to broaden the LLM's knowledge database. This approach offers a significant advantage avoiding spending resource on retraining LLMs. Instead it focuses on optimizing the utilization of existing models by equipping them with real-time access to external knowledge sources. Preliminary findings suggest this system has the potential to enhance LLM performance in knowledge-intensive tasks.

2.3.2.3 LLM Chatbot for Medicine

Another conversation agent using LLMs focusing on a specific domain is *CataractBot* [33]. CataractBot is an expert-in-the-loop chatbot system designed to provide useful information to patients undergoing cataract surgery. Developed in collaboration with Sankara Eye Hospital in India, CataractBot leverages a Large Language Model (LLM) alongside a curated knowledge base and expert's verification to provide accurate and trustworthy information to patients. The system is multilingual supporting five languages (English, Hindi, Kannada, Tamil, Telugu) and multimodal accepting both text and speech inputs with corresponding text and audio outputs. CataractBot prioritizes verified responses for medical questions by employing a combination of a custom knowledge database which includes hospital documents and FAQs and also utilizes expert's review by doctors to ensure accuracy of the information provided.

2.3.3 Grouping Discussion

One approach to categorize chatbots in general and more specifically in the cybersecurity field is by the underlying technology powering their functionality. This thesis proposes grouping chatbots based on their utilization of Transformers or Large Language Models (LLMs). This grouping focuses into the core functionalities and suitability of each type of chatbot for specific applications attempting to provide valueable insights of each technology.

Transformer-based chatbots are more suited on problems of identifying user intent and extracting key entities from their queries. This capability stems from the transformer architecture's ability to analyze complex relationships between words within a sentence. This makes them a natural fit for tasks like intent classification and Named Entity Recognition (NER) which are crucial for chatbots that search through vast databases for specific answers. For instance a transformer-based chatbot could effectively analyze a user query and identify the exact intent or recognize mentions of specific vulnerabilities so to provide optimal information.

LLMs use their vast training on massive text datasets to understand and generate human-like text. This advantage makes them ideal for tasks requiring comprehensive knowledge and the ability to communicate in a natural language. In the context of cybersecurity LLMs could be used to develop chatbots that provide users with detailed information of security threats and vulnerabilities.

It is important to acknowledge that this grouping is not mutually exclusive. Some chatbots may combine both transformer and LLM capabilities to achieve a broader range of functionalities. Future advancements in chatbot technology may further blur the lines between these categories creating chatbots that combine best features of the two worlds.

Chapter 3

Chatbot Database

3.1 Database Architecture

The proposed chatbot retrieves data from two different sources. The first one is a MISP (Malware Information Sharing Platform) SQL database. The MISP database provides access to a collection of malicious events, Indicators of Compromise (IoC) [34] and more, that would be more speficied on this chapter. The other source, are JSON files which contain a structured format of many documented vulnerabilities of different types. This combination allows the chatbot to make use of the extensive threat intelligence available through MISP while incorporating focused vulnerability information from the JSON files.

3.2 Sources of data

3.2.1 MISP Database Architecture

The MISP data model supports complex functionalities while at the same time follows a simple architecture. MISP employs a structured data model centered around the concept of an *event* object which is defined by a set of characteristics and descriptions [34]. The aforementioned characteristics are called *attributes* in this data structure and they focus into providing all useful information linked to an event. These attributes may include information about date, threat level, comments etc.

MISP's database schema supports managing complex threat data in a multi-user environment across various sectors. This is achieved through the implemented MISP attribute *categories, types, objects* and *taxonomies.*

Any Cyber Threat Intelligence (CTI) object is stored in the MISP database in the form of *attributes.* Multiple grouped attributes form an *object* which consists a bigger CTI artifact. Attributes and objects provide details for each event with attributes defining specific characteristics. MISP further allows correlation between events based on matching attributes, essentially creating connections that show shared characteristics as shown on Figure 3.1.

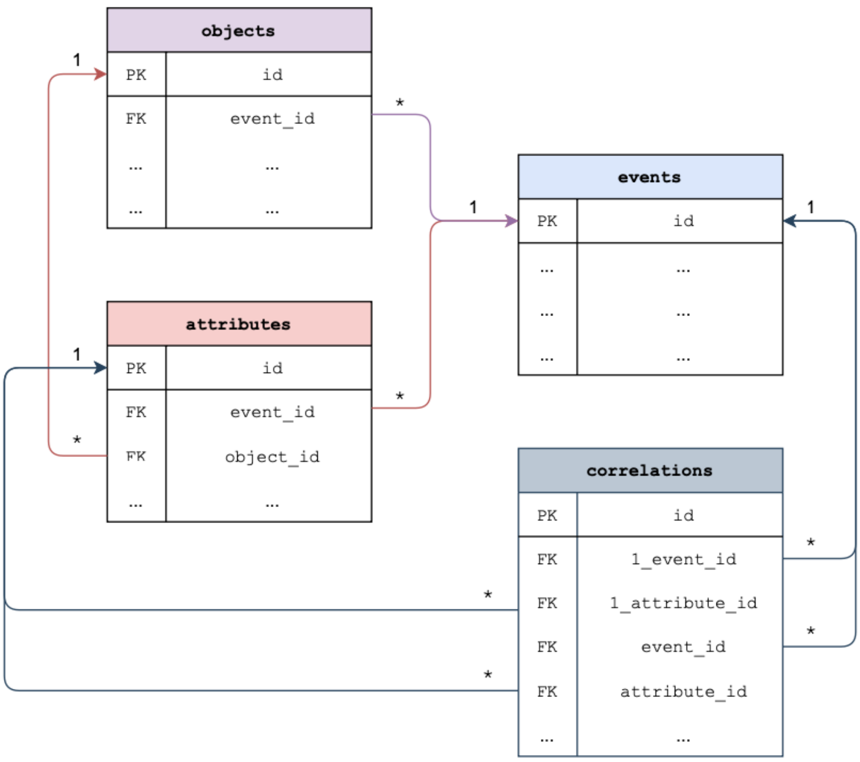


Figure 3.1: MISP correlations

3.2.1.1 Events Table

The events table acts as the core structure within MISP. It functions as a container containing attributes, objects, and metadata to create a comprehensive set of indicators describing a specific case. Events can represent various scenarios such as incidents, security analysis reports or threat actor analyses.

Some of the major fields in the events table are:

- the unique identifier of the event
- its publication status
- a threat level
- date
- the organisation which generated the event

and more information describing each instance.

3.2.1.2 Objects Table

Within an event, objects serve as containers that group related attributes. This allows describing more complex structures than what a single attribute can capture. Each object is built from a pre-defined template which provides a structure for the meta-data it carries. Objects are further categorized to a meta-category and defined by a name for easy identification.

Some of the major fields in the objects table are:

- the unique identifier of the object
- its name
- a meta-category
- description
- an event id the object belongs to

and many more informative fields.

3.2.1.3 Attributes Table

Attributes are the fundamental elements used to describe indicators and contextual data within an event. Each attribute consists of three key components a category, a type and a value which work together to provide meaning and context. The category and type give meaning and context to the value, while value itself holds the actual information. This combination allows MISP to represent a wide range of information.

Some of the fields worth mentioning are:

- the identifier attached to each attribute
- its type
- its predefined category
- event id referencing the event object this attribute belongs to
- object belongs to

and many more informative fields.

3.2.1.4 Correlations Table

Correlations table serve as a bonding system which provides connections between event objects. The main purpose of this table is to describe any correlations that may occurred between events through the MISP Correlation Engine.

Some worth mentioning fields of the correlation table are:

- the identifier
- the value which represents the payload of the correlated attribute
- l event id which represents the id of the event at hand
- event id which represents the event correlating to the one at hand

etc.

A comprehensive discussion of the aforementioned MISP tables, their corresponding fields and further information can be found within the MISP publication [34].

3.2.2 Vulnerabilities JSON Files

These vulnerability JSON files are containing many different instances of viruses, trojans, spywares and adwares. These data were provided by the thesis' supervisor Dr. Christos Tryfonopoulos. The fields these collections contain are presented in Tables 3.1 - 3.4 separated for each vulnerability category.

Fields	Explanation
id	Unique identifier of virus entry
name	Name of the virus
type	Type of vulnerability
date	Discovery date of virus
url	Website with more information
infection length	Size of the virus
systems affected	The operating system the virus can infect
risk impact	Severity of the virus
also known as	Potential alternative names of the virus
antivirus protection dates	First detection and the last update to the detection mechanism.
technical description	Describes how the virus works and what it does to a computer system.
recommendations	General recommendations for preventing infection.
removal	Instructions for removing the virus, typically using Symantec's antivirus products.

Table 3.1: Virus Collection

Fields	Explanation
id	Unique identifier of trojan entry
name	Name of the trojan
type	Type of vulnerability
date	Discovery date of trojan
url	Website for further information
discovered	Discovery date written in full
infection length	Size of the trojan
systems affected	The operating system the trojan can infect
installation	How the trojan is installed on a system
permissions	Permissions requested by the trojan during installation
functionality	What the Trojan does on a compromised device
recommendations	Recommendations for preventing infection or mitigating risks
removal	Instructions for removing the trojan

Table 3.2: Trojan Collection

Fields	Explanation
id	Unique identifier of Spyware entry
name	Name of the Spyware
type	Type of vulnerability
date	Discovery date of Spyware
url	Website for further information
infection length	Size of the Spyware
systems affected	The operating system the Spyware can infect
behavior	What the Spyware does on a compromised device
antivirus protection dates	Dates the antivirus software recognizes this Spyware
installation	How the Spyware is installed
permissions	Permissions requested by the Spyware during installation
functionality	What the Spyware does on a compromised device
removal	Instructions for removing the Spyware

Table 3.3: Spyware Collection

Fields	Explanation
id	Unique identifier of Adware entry
name	Name of the Adware
type	Type of vulnerability
date	Discovery date of Adware
url	Website for further information
infection length	Size of the Spyware
version	Version (if applicable)
publisher	Organisation publishing it
risk impact	Severity of the Adware
behavior	What the Adware does on a compromised device
systems affected	Affected operating systems
antivirus protection dates	Dates the antivirus software recognizes this Adware
technical description	What the Adware does on a compromised device
removal	Instructions for removing the Spyware

Table 3.4: Adware Collection

3.3 Data exploration and statistics

After acquiring the data from both the MISP database and the vulnerability JSON files, Python scripts were employed to conduct an exploratory analysis. This analysis focused on the distribution of data fields of the dataset attempting to identify informative fields that could also be leveraged for the development of a more expansive chatbot.

Integrating in the chatbot fields that exhibit variability across diverse vulnerability scenarios enhances the informativeness of the conversations. On the other hand fields with constant values or lacking content (null) provide no significant value. The exclusion of such fields has led to more insightful chatbot dialogues.

For example below are presented two samples of the plots created for various fields of the data showing the distribution of their values.

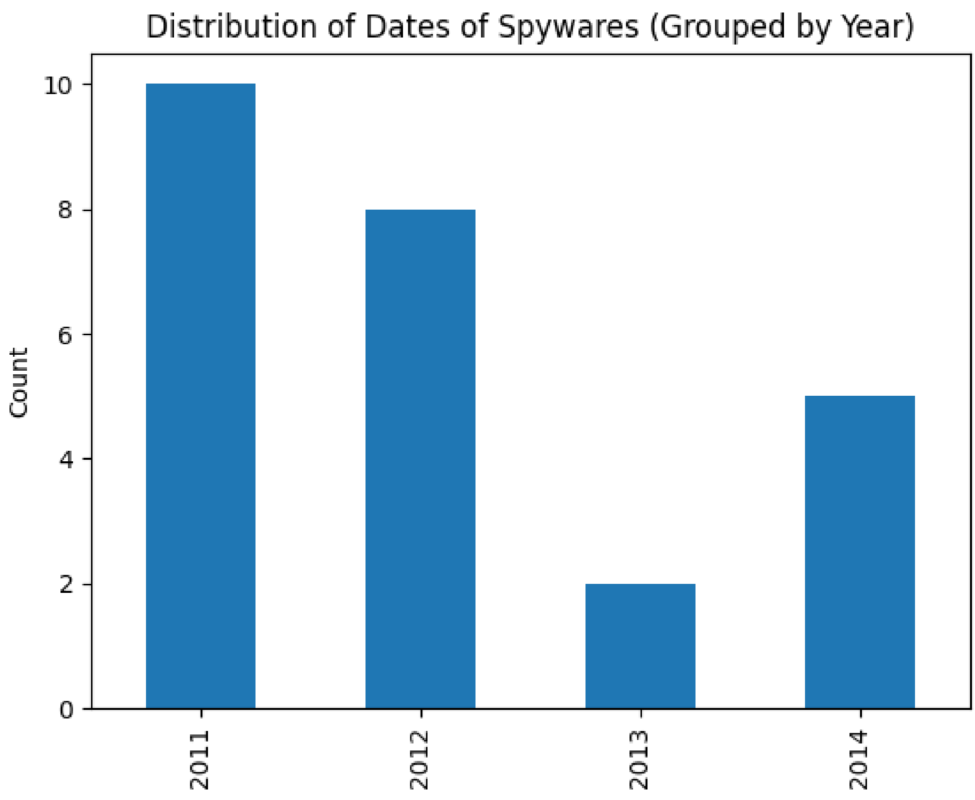


Figure 3.2: Spyware Dates Distribution

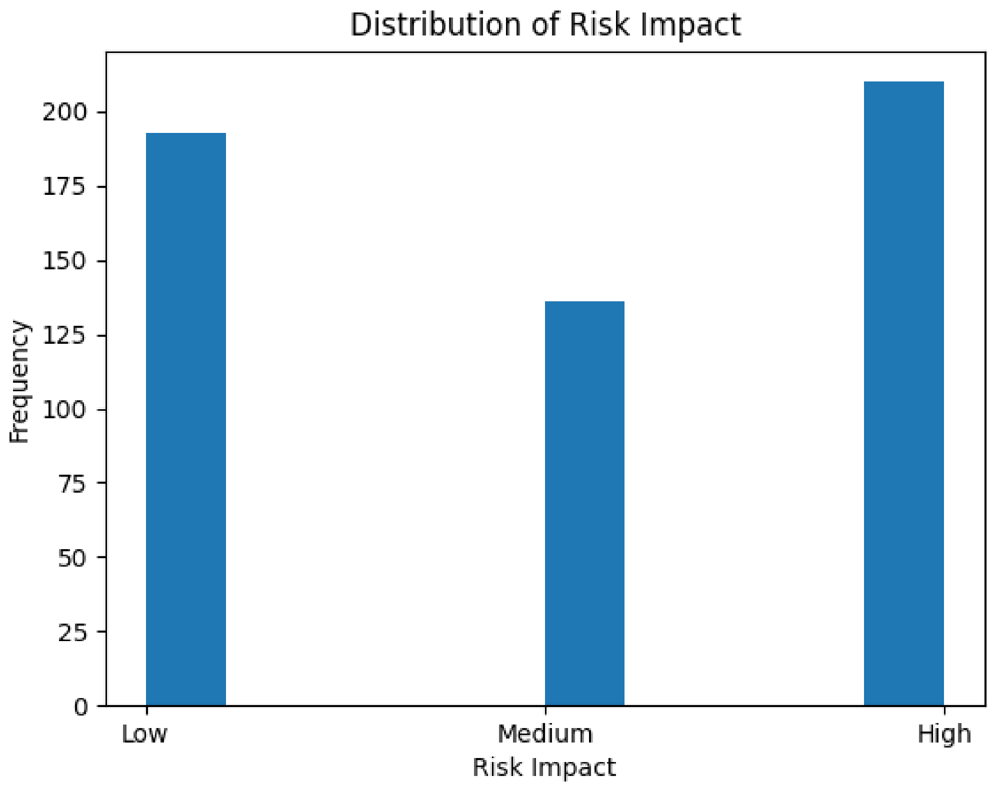


Figure 3.3: Adware Risk Impact Distribution

Chapter 4

Chatbot Architecture

This chapter presents the architectural choices made for the proposed chatbot. Since this chatbot is based on the Rasa framework, a detailed description of its architecture will be provided in the following sections. These sections will also comprehensively explore the way the proposed chatbot utilizes the capabilities offered by Rasa.

4.1 Rasa Framework

Rasa [29] is a powerful open-source framework for creating chatbots and conversational AI assistants. Rasa uses machine learning and Python allowing you to understand, hold conversations and connect to messaging channels and third party systems through a set of APIs.

Rasa Conversational AI assistant normally consists of two components, Rasa NLU and Rasa Core. Rasa NLU can be treated like a receiver which processes the incoming messages from the user handling intent classification, entity extraction, and response retrieval. Rasa Core handles the dialog management process taking decisions based on user input.

4.2 Rasa Architecture

This section describes the architecture of the Rasa framework, providing an extensive presentation of its fundamental components. The description of these various components will attempt to provide insights of the mechanisms that enable chatbots based on Rasa to comprehend user queries and generate appropriate responses.

4.2.1 Overview of Rasa Architecture

Rasa relies on dialogue policies to determine its responses. The aim of these policies is to analyze the conversation state which is stored in the tracker object and to choose the most suitable action to take next. A pipeline of how this exactly works follows:

- Rasa NLU component processes a user input so to identify the intent and extract its entities.
- Rasa keeps track of the dialogue state by storing the user's previous inputs, the system's responses, all the extracted entities and many other metadata.
- Rasa's dialogue policy is responsible to decide the next action based on the current dialogue state. This decision can either be rule-based where predefined rules match specific conditions or based on machine learning where a model predicts the most suitable action
- After all previous steps have been made Rasa generates the response and sends it to the user as a reply.

4.2.2 Rasa Components

This section attempts to present the key components of Rasa, including intent recognition (IR), entity extraction, tokenizers and featurizers. Utilizing these components, the chatbot is able to discover the intention of user queries, extract the useful information of the provided message and then convert it to numerical vectors for further processing.

4.2.2.1 Intent Recognition

Intent recognition (IR) is the component responsible for predicting the user's intent depending on the input. IR analyzes the user's message and categorizes it into labels, called intents. Rasa NLU will classify the user messages into one or also multiple user intents. The two components between which you can choose are:

- Pretrained Embeddings (Intent classifier sklearn)
- Supervised Embeddings (Intent classifier tensorflow embedding)

The Sklearn [35] classifier uses the spaCy [36] library to load pretrained language models to convert each word into a numerical vector (word embedding). Word embeddings are vector representations of words, meaning each word is converted to a dense numeric vector. Word embeddings capture semantic and syntactic aspects of words. This means that similar words should be represented by similar vectors.

Rasa NLU takes the average of all word embeddings within a message and performs a grid search to find the best parameters for the support vector classifier which classifies the averaged embeddings into the different intents. The grid search trains multiple support vector classifiers with different parameter configurations and then selects the best configuration.

The Tensorflow [37] embedding intent classifier was developed by Rasa and is inspired by Facebook's starspace paper [38]. Instead of using pretrained embeddings and training a classifier on top of that, Rasa offers an alternative approach which trains word embeddings from scratch. It is typically used with the intent featurizer count vectors component which counts how often distinct words of your training data appear in a message and provides that as input for the intent classifier.

Intent classification is an essential element of natural language understanding (NLU) in chatbot development. The predicted intent triggers specific actions routing the conversation flow. Intent classification is stated through a configurable pipeline. This pipeline is typically defined within a configuration file *config.yml*. The optimal selection of the intent classifier is dependent on the chatbot design and the characteristics of the training data. Rasa provides default configurations for a variety of intent classification algorithms but configuring these algorithms for each specific case can help a lot its performance.

4.2.2.2 Entity Extraction

Entity extractors are responsible to identify and extract specific information (entities) from user messages. These entities represent data points that the chatbot uses during conversation handling so to produce targeted responses.

The training data provided are of high significance in this process. By analyzing this data, entity extractors learn to understand patterns between words and their entity labels. Implementing that, entity extractors can then accurately recognize and extract entities from novel user messages. The extracted entities are responsible for populating slots or variables which function as memory locations for storing information during of the conversation.

4.2.2.3 Tokenizers

Tokenizers are a fundamental component of Natural Language Process (NLP) responsible for breaking text input into distinct units referred to as tokens. These tokens represent the most basic units that are deemed non-decomposable for subsequent processing stages. Tokenization enables the system to process and understand text at a higher level. Tokenizers split text input into segments, typically at the word level, while separating individual words, punctuation marks, and other meaningful units in the text. For languages that are not dependent on whitespaces such as Chinese there are different tokenizers.

In Rasa, the most used tokenizers a:

- WhitespaceTokenizer: This tokenizer splits the text based on whitespace characters
- SpacyTokenizer: Implementation of the SpaCy tokenizer
- MitieTokenizer: Implementation of MIT's Information Extraction tokenizer
- JiebaTokenizer: Tokenizer using Jieba for Chinese language

4.2.2.4 Featurizers

Featurizers are components converting raw textual user inputs into numerical feature vectors that are being processed by machine learning models. These vectors capture important information from the input which help in predicting the user's intent or extracting entities.

Text featurizers are divided into two different categories: sparse featurizers and dense featurizers. Sparse featurizers are featurizers that return feature vectors with a lot of missing values, e.g. zeros. As those feature vectors would normally take up a lot of memory, we store them as sparse features. Sparse features only store the values that are non zero and their positions in the vector. Thus, we save a lot of memory and are able to train on larger datasets.

All featurizers can return two different kind of features: sequence features and sentence features. Sequence features are a matrix which contains a feature vector for every token in the sequence allowing to train sequence models. The sentence features are represented by a matrix which contains the feature vector for the complete utterance. The sentence features can be used in any bag-of-words model.

Rasa supports many featurizers either using existing ones like MITIE, ConveRT and SpaCy or custom using Regular Expressions (RegexFeaturizer) and pretrained language models (LanguageModelFeaturizer) [39].

4.3 Introduction To The Implemented Chatbot

The chatbot implemented in this study is designed to provide users with quick and accurate information about cybersecurity threats and vulnerabilities (CVEs, viruses, trojans, etc.). The proposed chatbot is designed to have also general purpose conversations like greetings and basic discussion to be more immersive. Currently the chatbot is designed to support the English language. In what follows, we provide a detailed description of the architecture and all different aspects of the proposed chatbot.

4.3.1 The NLU Structure of the chatbot

The proposed chatbot starts its flow by receiving the input from the user, as seen in Figure 4.1. The first step after getting the user's input is to break it into distinct tokens. For that purpose the built-in Rasa's **WhitespaceTokenizer** [40] was firstly utilized so to break the provided sentence(s) into words, as the most common and suitable for the English language. WhitespaceTokenizer is based only on splitting text based on whitespace characters. In the journey of finding the most suitable for this case, we tried multiple tokenizers but the best results were provided by **spaCyTokenizer**. The SpaCy Tokenizer was used over WhitespaceTokenizer, as spaCy is a powerful NLP library performing additionally stemming/lemmatization over WhitespaceTokenizer. The results in intent prediction distribution were significantly better than WhitespaceTokenizer reducing the wrongly predicted intents by 20%. Despite the benefits of the usage of spaCyTokenizer, the increase in training time was also significant. Since the implemented chatbot does not deal with massive datasets yet, spaCy tokenizer was selected as a suitable fit. If performance becomes an issue, switching to faster WhitespaceTokenizer should be operated.

This choice had to be done, as Rasa does not allow for multiple tokenizers due to the sequential process of text. This is not the case in entity extractors, as multiple extractors can improve accuracy and flexibility of entity recognition. In the proposed chatbot, **CRFEntityExtractor**, **SpacyEntityExtractor** and **RegexEntityExtractor** are used. Regex Entity Extractor is simple and handles well entities described by regular expressions, making it suitable for this chatbots' needs. In the context of the proposed chatbot, these patterns can be the names of the cyberthreats or dates of events. Also, SpaCy Entity Extractor is a great match because spaCy tokenizer is also present in the model. These two components collaborate very well with each other, and also this extractor can leverage spaCy's pre-trained models for entity extraction. These two are combined with the CRF entity extractor, which can learn complex patterns for entity recognition and handling overlapping entities. This combination proved to provide great results in entity recognition (Figure 4.8).

Other useful component implemented in the proposed chatbot's configuration file, is the **EntitySynonymMapper**. This component checks if the recognized entities have any synonyms in the declared in Rasa NLU.yml file. By recognizing synonyms of entities, the model can better interpret the intent even the user does not use any of the exact words documented in the NLU file.

After that, the extracted words are becoming numerical vectors by the procedure previously described in section 4.2.2.4 as **featurization**. For that task the **CountVectorsFeaturizer** was utilized as the most naive solution, since it works on the principle of word frequency, it analyzes the training data and learns how often each word appears and creates a feature for each word. CountVectorsFeaturizer creates features for intent classification using bag-of-words reprsentation of user message. As Rasa allows more than one featurizers in the configuration, **SpaCyFeaturizer** was also used (Figure 4.5) which matches very well the pre-trained SpaCy's language models, and is a good practice to be implemented combined. SpaCy is feature rich and can effectively extract a rich set of features from the tokenized text, including part-of-speech tags and word embeddings. After careful consideration and some tests, in the final featurizer setup **ConveRTFeaturizer** was also added as it added a small increase in intent recognition. The advantage of the ConveRTFeaturizer is that it doesn't treat each word of the user message independently, but creates a contextual vector representation for the complete sentence.

The most crucial part in the Natural Language Understanding (NLU) is definitely the **intent classification**. If the chatbot manages to correctly identify the intentions of the given input then it can accordingly provide a suitable response. For the purpose of the proposed chatbot, **DIET classifier** was selected as the most optimal tool for the current data. DIET classifier [41] is a common choice for similar tasks and is also used on [27] as described on Chapter 2.1.1.1.

The **Dual Intent and Entity Transformer** (DIET) [41] is a transformer-based architecture specifically designed for Rasa and was introduced on Rasa 1.8 version in 2020. As its name suggests, DIET can handle both intent classification and entity recognition. A key strength of DIET is its flexibility. It offers the capability to integrate various pre-trained word embedding models, such as BERT, GloVe, and ConveRT. These pre-trained embeddings are representing words as numerical vectors, facilitating the process of intent classification and entity recognition.

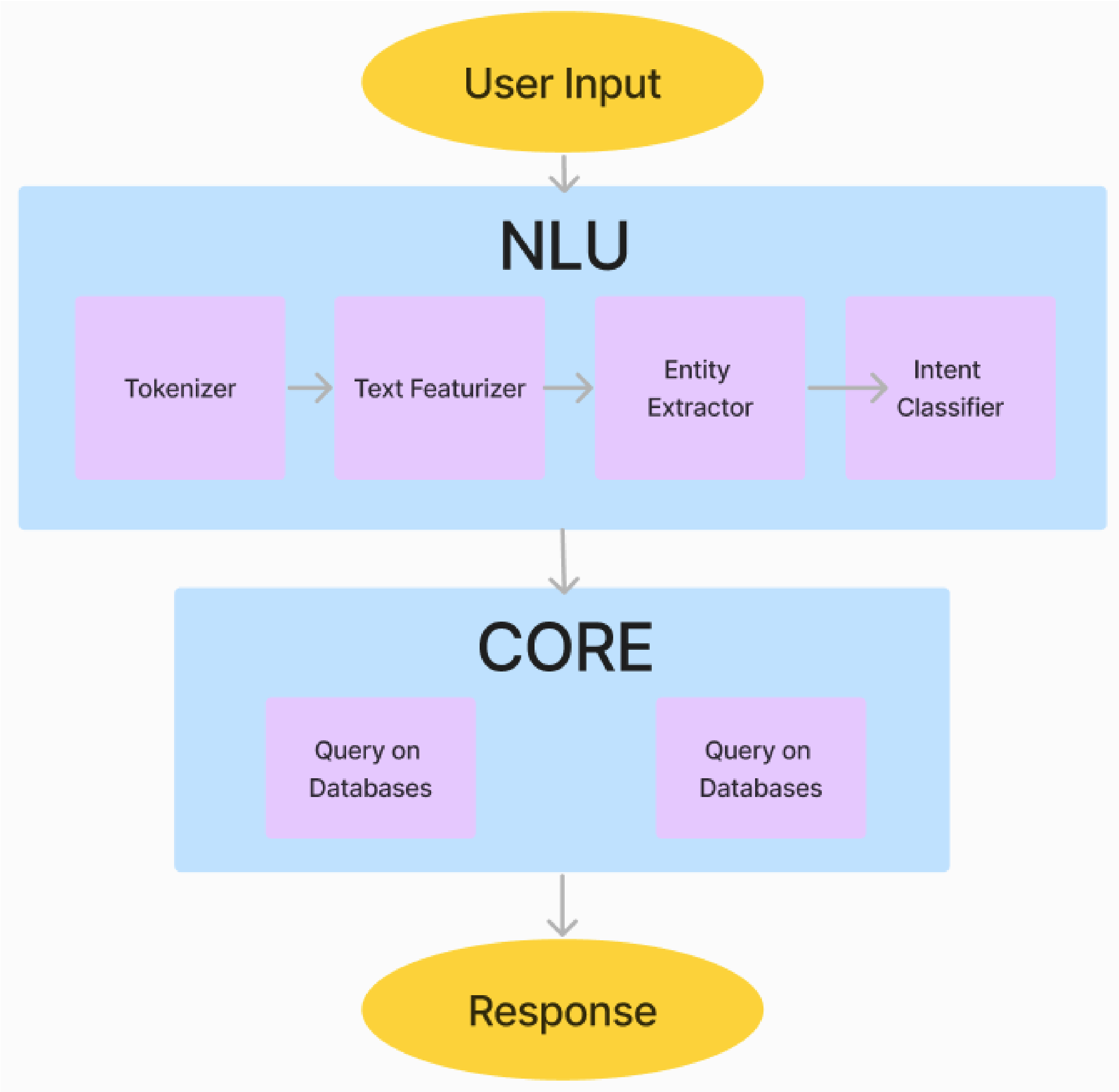


Figure 4.1: Chatbot Flowchart

4.3.2 Chatbot's Rasa Stack

4.3.2.1 Nlu.yml

Within the NLU.yml file, all user intents are documented along with a sample of possible questions that users may use while expressing their desire. These modeled sentences constitute the training data of our chatbot, meaning that users that express the same intent with different ways will be served of the intended response.

Below follows Table 4.2 containing all different intentions and the purpose they deliver.

Intent	Purpose
how are you	Responds to greetings like hello, hi, etc.
fun fact	Shares a fun fact with the user.
thanks	Acknowledges the user's thanks.
bot challenge	Identifies when a user is checking if they are interacting with a bot.
bad	Handles negative feedback from the user.
good evening	Responds to greetings specific to evening time.
good afternoon	Responds to greetings specific to afternoon time.
good morning	Responds to greetings specific to morning time.
wrong	Identifies when a user disagrees with the response.
bye	Responds to goodbyes from the user.
whats up	Engages in small talk with the user.
misp describe event	Queries MISP for event details based on a provided ID.
misp describe object event	Queries MISP for object details based on a provided event ID.
misp describe previous event references	Retrieves references for a previously discussed event.
misp describe previous event date	Retrieves the date of a previously discussed event.

misp describe threat levels	Retrieves threat levels available in MISP.
misp print events threat level	Counts events based on the specified threat level.
misp list tables	Lists all the tables in the MISP database.
misp print latest earliest table row	Retrieves the latest or earliest entry from a specified MISP table.
misp count table rows	Counts the number of entries in a specified MISP table.
misp select table	Selects entries from a MISP table based on various filters (e.g., get me all correlations from the events table).
get trojan functionality by name	Provides information about a trojan's functionality based on its name.
trojan discovery date	Asks for the discovery date of a trojan.
trojan installation	Asks for the installation procedure of a trojan.
trojan permissions	Asks for the permissions of a trojan.
trojan recommendations	Asks for recommendations regarding a trojan.
trojan url	Asks for the URL linked to a trojan.
get spyware functionality by name	Provides information about a spyware's functionality based on its name.
spyware risk impact	Asks for the risk impact of a spyware.
spyware installation	Asks for the installation procedure of a spyware.
spyware url	Asks for the URL linked to a spyware.

spyware discovery date	Asks for the discovery date of a spyware.
get virus description by name	Provides information about a virus's description based on its name.
virus url	Asks for the URL linked to a virus.
virus discovery date	Asks for the discovery date of a virus.
virus protection date	Asks for the protection date of a virus (when it was solved).
virus recommendations	Provides recommendations for dealing with a virus.
get adware description by name	Provides information about an adware's behavior based on its name.
adware url	Asks for the URL linked to an adware.
adware risk impact	Asks for the risk impact of an adware.
adware protection date	Asks for the protection date of an adware.
adware discovery date	Asks for the discovery date of an adware.
get vulnerability type by name	Identifies the type of vulnerability based on its name.
count spywares by risk impact	Calculates the number of spywares with a specified risk impact.
vulnerability url	Asks for the URL linked to a vulnerability.
count threats after date	Calculates the number of threats after a specified date.
count threats before date	Calculates the number of threats before a specified date.

count threats by risk impact	Calculates the number of threats with a specified risk impact.
count threats by risk impact and after date	Calculates the number of threats with a specified risk impact after a specified date.
count threats by risk impact and before date	Calculates the number of threats with a specified risk impact before a specified date.
get latest threat based on risk impact	Retrieves the latest threat with a specified risk impact.

Table 4.2: Summary of Bot Intents and Purposes

In Figure 4.2, follows a brief snapshot showing how actually this file is structured. For each intent numerous examples of user queries are documented. The creator of this file cannot foretell all possible user questions regarding each intent, but providing the chatbot many different approaches of questions for an intent helps a lot the intent prediction process. This helps the model to generalize better and accurately classify new user inputs, even if phrased differently from the training data. On that task EntitySynonymMapper component, as described in 4.3.1, helps a lot when the model meets a word that is not documented in the NLU.yml file but characterizes one intent. A larger dataset also helps the model understand the differences between intents which reduces the chances of it falsely classifying an intent.

In the same Figure 4.2, it is observed that while on *virus_recommendations* intent the examples are simple questions, on the *get_adware_description_by_name* intent, the entity *adware_id* is included. This means that this intent expects the adware_id entity to be present into the user input. To declare that in the NLU.yml file, first it is required to declare the regular expression that the entity must match. After that, in the intent examples it is declared by providing an adware_id example inside brackets (e.g. [Adware.Jogotempo]) followed by the name of the entity inside parentheses (e.g. (adware_id)).

```
- intent: virus_recommendations
  examples: |
    - What are the recommendations for that virus?
    - Virus recommendations
    - Give recommendations for this virus
    - Provide recommendations regarding the previous virus
    - And what about this virus' recommendations?
    - Any recommendations for that virus?

- intent: get_adware_description_by_name
  examples: |
    - Provide [Adware.Jogotempo](adware_id) adware behavior
    - Get me [Adware.FreeVideoConvtr](adware_id) adware behavior
    - Provide adware [Adware.IstartSurf!lnk](adware_id) functionality
    - Get me adware [Adware.GoSave](adware_id) functionality
    - Provide info about adware [Adware.MusicDowector](adware_id)
    - Describe adware [Adware.Social2Search](adware_id)
    - Do you have any information about adware [Adware.BetterSurf](adware_id)?
    - Do you have a technical description for adware [Adware.ZombieInvasion](adware_id)
    - Give technical description of [Adware.SearchAssist](adware_id) adware
    - Provide [Adware.Browext](adware_id) adware technical description
```

Figure 4.2: The NLU.yml file

4.3.2.2 Stories.yml

This file provides all potential conversational interactions between a user and a chatbot. Each user intent is accompanied by a corresponding action enabling the chatbot to generate its dialogues. The role of the stories in the conversation flow is crucial as it matches intents with the desired response actions. In the proposed chatbot, stories follow a simple format, as also seen in Figure 4.3, where intents and actions are coupled together within a story.

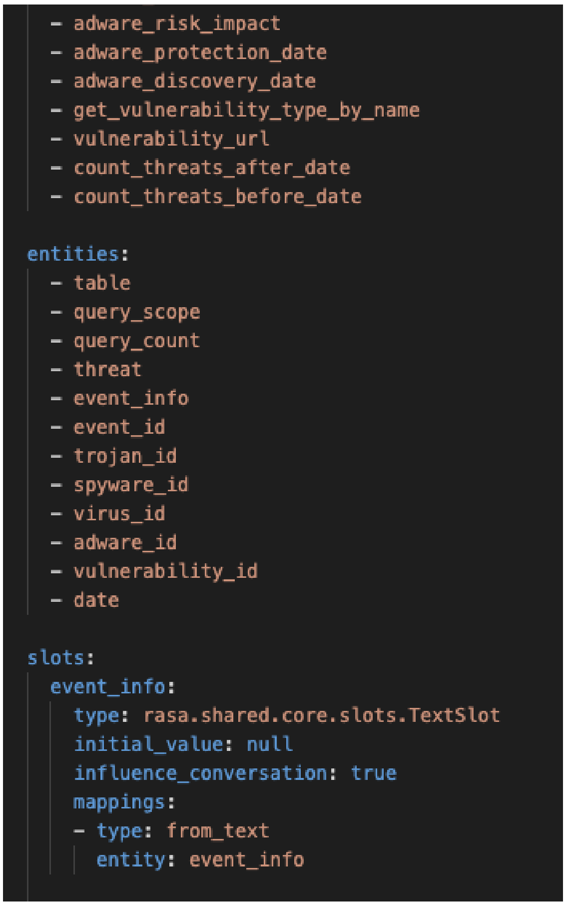
```yaml
- story: Count how many threats based on risk impact before date
  steps:
  - intent: count_threats_by_risk_impact_and_before_date
  - action: action_count_threats_by_risk_impact_and_before_date

- story: Provide the most recent threat based on risk impact
  steps:
  - intent: get_latest_threat_based_on_risk_impact
  - action: action_get_latest_threat_based_on_risk_impact

- story: How are you
  steps:
  - intent: how_are_you
  - action: utter_how_are_you
```

Figure 4.3: The STORIES.yml file

4.3.2.3 Domain.yml

The domain.yml file defines the universe in which the conversational assistant operates. It specifies the intents, entities, slots, responses, forms, and actions the bot should know about. It also defines a configuration for conversation sessions. A part of it is described in Figure 4.4. The domain.yml file acts mostly as a declaration reference for all different aspects of the chatbot.

While entities and intents have been discussed in section 4.3.2.1, *slots* is a new term found on the domain.yml file. Slots act as a key-value store which can be used for storing user provided information. In Figure 4.4 a slot is defined with name "event_info", type TextSlot (for storing textual data) and predefined slot mapping from_text (filling the slot based on text patterns). Also, the *influence_conversation* parameter is set to true as it is essential for entities to influence the conversation flow. Otherwise, the model will keep the value of the entity but will not assist the conversation flow with its value.



```
  - adware_risk_impact
  - adware_protection_date
  - adware_discovery_date
  - get_vulnerability_type_by_name
  - vulnerability_url
  - count_threats_after_date
  - count_threats_before_date

entities:
  - table
  - query_scope
  - query_count
  - threat
  - event_info
  - event_id
  - trojan_id
  - spyware_id
  - virus_id
  - adware_id
  - vulnerability_id
  - date

slots:
  event_info:
    type: rasa.shared.core.slots.TextSlot
    initial_value: null
    influence_conversation: true
    mappings:
    - type: from_text
      entity: event_info
```

Figure 4.4: The DOMAIN.yml file

4.3.2.4 Config.yml

In this file certain policies and pipelines for the efficiency of the Rasa NLU and the Rasa Core are defined. Here, the desired Rasa components (e.g, Intent Classifier, Entity Extractor, Epochs etc.) are declared for that chatbot. During the training process, attributes from this file were changed accordingly to achieve improved accuracy and F1-score.

The config.yml file of the proposed chatbot took its final form presented in Figure 4.5 after a long trial and error process. This procedure included a lot of reading in the Rasa documentation [39] so to identify the most suitable pool of featurizers, tokenizers, entity extractors and classifiers for the needs of this specific chatbot. After selecting the suitable components for our case, many combinations of them, with different configuration took place trying to find the optimal configuration setup.

The fundamental components regarding the tokenizer, featurizers, entity extractor, synonym mapper and the classifier found in the CONFIG.yml fo Figure 4.5 were thouroughly described combined with the thought process behind their selection in section 4.3.1. Present in the pipeline section of CONFIG.yml file is also the **FallbackClassifier**. This component handles incoming messages with low NLU confidence. The threshold seen on Figure 4.5 under FallbackClassifier indicates that all messages getting under that confidence percentage will fall under the fallback category. More precisely, this requires a new rule to be added on the model, indicating that all user messages with fallback intent will response with a specified action.

All these components are included in the *pipeline* section of the configuration file. The configuration file of Rasa has two parts, the *pipeline* and the *policies*. The pipeline specifies the components used by the model to make NLU predictions. The policies on the other hand, define the policies used by the model to predict the next action. The components used in the policies for the proposed chatbot are the most fundamental ones, fine tuned to balance accuracy and training time whenever was necessary.

The first component is the **MemoizationPolicy** which remembers the stories from the training data. It checks if the current conversation matches the stories in stories.yml file. If so, it will predict the next action from the matching stories of your training data with a confidence of 1.0. If no matching conversation is found, the policy predicts None with confidence 0.0. **RulePolicy** is another component, which couples the previous naive policy. RulePolicy is a policy that handles conversation parts that follow a fixed behavior. It makes predictions based on any rules that exist in the rules.yml file. As described previously in the same section, one example of rule is in the fallback scenario in which a specific answer is given. For more information in rules.yml file please see on section 4.3.2.5.

One more policy component used is the UnexpecTEDIntentPolicy which is designed to help identify and address unexpected user inputs. It analyzes conversation history and assesses the likelihood of the most recent user intent based on the training data. If the intent is flagged highly improbable, it triggers a special action indicating an unexpected input. This policy aids in identifying areas where training data is insufficient and highlights potentially problematic interactions for review. The max_history parameter (here set to 5) controls the number of user utterances the model will look back when evaluating the likelihood of the most recent user intent.

The last policy used is the **TEDPolicy** which is a multi-task architecture for next action prediction and entity recognition. The architecture consists of several transformer encoders which are shared for both tasks. A sequence of entity labels is predicted through a Conditional Random Field (CRF) tagging layer on top of the user sequence transformer encoder output corresponding to the input sequence of tokens. For the next action prediction, the dialogue transformer encoder output and the system action labels are embedded into a single semantic vector space. TEDPolicy uses the dot-product loss to maximize the similarity with the target label and minimize similarities with negative samples.

Both UnexpecTEDIntentPolicy and TEDPolicy are very dependent on the training data. For that reason the epochs parameter is set for both policies to 100 instead of the default value of 1. This means that these policies will pass the training data 100 times back and forth. With a single epoch, the model might not have learned everything it can from the data. By specifying a higher number of epochs (e.g., 100), the model is allowed to iterate through the training data multiple times. Each time, it refines its understanding based on the accumulated knowledge from previous passes. To determine the number of epochs for these policies a lot of testing was held having in mind both intent accuracy and training time. 100 epochs added a 10% overhead in the training processing time (around 2 minutes) but also assisted in the intent prediction by giving almost 100% confidence to more examples.

```yaml
language: en

pipeline:
  - name: SpacyTokenizer
  - name: CRFEntityExtractor
  - name: RegexEntityExtractor
  - name: SpacyEntityExtractor
  - name: EntitySynonymMapper
  - name: CountVectorsFeaturizer
  - name: SpacyFeaturizer
  - name: ConveRTFeaturizer
  - name: DIETClassifier
  - name: FallbackClassifier
    threshold: 0.4

# Configuration for Rasa Core.
# https://rasa.com/docs/rasa/core/policies/
policies:
  - name: MemoizationPolicy
  - name: RulePolicy
  - name: UnexpecTEDIntentPolicy
    max_history: 5
    epochs: 100
  - name: TEDPolicy
    max_history: 5
    epochs: 100
    constrain_similarities: true
assistant_id: 20231207-212416-tangy-foie-gras
```

Figure 4.5: The CONFIG.yml file

4.3.2.5 Rules.yml

In Rasa, rules are a type of training data used to train the dialogue management model. They describe short, predictable interactions that should always follow the same path. In this case, rules file only contain the fallback intent. Fallback intent is triggered when the NLU confidence for a specific user message is under the threshold defined in the config.yml file (section 4.3.2.4). If this fallback is triggered, the chatbot follows the corresponding rule shown in Figure 4.6, to ask kindly the user to rephrase.

```yaml
version: "3.1"

rules:
- rule: Ask the user to rephrase whenever they sendd a message with low NLU confidence
  steps:
  - intent: nlu_fallback
  - action: utter_please_rephrase
```

Figure 4.6: The RULES.yml file

4.3.2.6 Actions.py

The file actions.py is in Python programming language and serves as the code base for custom actions. These actions are referenced in the aforementioned configuration files and are matched with the corresponding user intent. The code within actions.py executes as intended by the developer fulfilling the desired functionality for each action. For complex actions, excluding greetings, the current implementation relies on custom Python classes. This approach makes it easier to handle new types of actions as the chatbot grows in size while also improves code organization and maintainability. That code is firstly responsible for getting the entities from Rasa's entity repository. Then it forms the queries for the database and serves the response.

In Figure 4.7 the MispQueryTableRows action is presented as an example. The three important components of the action file, present in all actions, are the **dispatcher**, the **tracker** and the **domain**. The dispatcher object is responsible to send messages back to the user using its *utter_message* method. Dispatcher can send text, images and even interactive buttons. The tracker object keeps track of the conversation history enabling the action to access current and previous entities, intents, messages and more from the user messages. The domain object contains information about all actions, entities, intents and slots the chatbot is able to use.

```python
class MispQueryTableRows(Action):
    def name(self) -> Text:
        return "action_misp_count_table_rows"

    def run(self, dispatcher: CollectingDispatcher,
            tracker: Tracker,
            domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
            # get table
            table = next(tracker.get_latest_entity_values("table"), None)
            # get query count (how many, a lot of, count, lots)
            query_count = next(tracker.get_latest_entity_values("query_count"), None)

            # initialize connection
            conn = DbQueryingMethods.create_connection(dataBaseFile="cmisp.db")

            # if table does not exist reply to user to enter a valid table
            mispTables = pd.read_sql_query("SELECT name FROM sqlite_master WHERE type='table';", conn)

            if DbQueryValidations.validateTableName(conn, table) == False:
                dispatcher.utter_message(text = str("Please enter a valid table name. Type 'MISP tables' to get a list of the available tables."))
                return []

            # if query count is not set reply to user to clarify
            if query_count is None:
                dispatcher.utter_message(text = str(f"Please clarify whether you want me to count all rows in {table}."))
                return []

            # finally proceed to count the table rows
            result = DbQueryingMethods.count_table_rows(conn, table)
            dispatcher.utter_message(text = str(f"Table {table} has {result} records."))
```

Figure 4.7: The ACTIONS.py file

4.3.3 Human Text Translation To Database Query

The design and selection of possible dialogues and the corresponding intents involved an exploratory data analysis process. This process focused on searching and querying the SQL database and the vulnerability JSON files for interesting insights and meaningful combinations between tables or fields. The SQL and Python queries that offered interesting results were documented and grouped. Also the necessary entities to facilitate the variability of the queries were specified based on the threats or events users might be interested into.

The documentation of queries and entities proved very important during the dialogue creation process. This documentation helped a lot in the formulation of questions and corresponding responses that directly addressed the documented queries.

4.3.4 Messaging Application Integration

This section explores how Rasa can deploy a chatbot into popular messaging applications. In the following section, the establishment of a publicly accessible URL for the implemented chatbot is described, as well as integrating it with popular messaging platforms.

4.3.4.1 Chatbot Tunneling Deployment

Rasa provides functionalities that enable the integration of the developed chatbot into the web with a publicly accessible URL. This integration facilitates the exchange of messages between the chatbot and users via messaging applications.

To establish a publicly accessible URL for the locally running chatbot the software **Ngrok** was used. Ngrok is a famous cross-platform application that creates secure tunnels (paths) to localhost machine. It enables developers to expose a local development server to the Internet. The software makes your locally-hosted web server appear to be hosted on a subdomain of ngrok.com, meaning that no public IP or domain name on the local machine is needed.

4.3.4.2 Connection to Slack

When the publicly accessible URL of the chatbot is established and available, the next step involves creating an application within the Slack settings. On that newly created application the url is declared to enable the connection.

The final step in this process includes copying private *token* and *signing secret code*, generated from the newly created application, into the **credentials.yml** file within the Rasa project. That way a successful connection is established and the chatbot is able to run on Slack. It is important to note that Slack is not the only messaging application supported. The framework supports many others like Facebook's Messenger and Mattermost.

4.3.5 Integration On Other Systems

The complete codebase for this chatbot is available for download on GitHub, as well as the SQL database and the threats/vulnerabilities JSON files. In addition, a README file is provided on GitHub. This file offers detailed instructions on setting up this chatbot on different systems.

This chatbot is implemented on a **Linux** operating system on **Python 3.10** programming language and **Rasa version 2** or higher. However, Rasa is flexible and can be configured and deployed also on Windows and Mac operating systems. With proper configuration the chatbot is expected to run unproblematically across these operating systems.

4.4 Chatbot Evaluation

The safest way to test a chatbot's functionality is by creating exhaustive scenarios to test its behavior and how close it is to its expected output.

4.4.1 Accuracy Metrics

This could be done by using the Rasa's built-in test capabilities. The way this works is by creating scenarios stating the input and the output and checking how close the real chatbot's response is. To do that, the data are split into test and train data with the command *rasa test nlu –nlu train_test_split/test_data.yml*. For better results cross-validation technique could be utilized with the addition of –*cross-validation* in the previous command. Cross-validation automatically creates multiple train/test splits and averages the results of evaluations on each train/test split. For the testing purposes of this chatbot, cross-validation was used with 5 folds. 5 folds is a common choice as it offers a good balance between accuracy estimation and efficiency without making the process excessively time-consuming given also that the training data are not large.

This procedure has as an output several plots providing insights about the chatbots performance on accuracy. These plots contain confusion matrices for all supported intents and for the DIET classifier based on the entities. Also histograms are produced showcasing the confidence level for both intents and entities.

The confusion matrix in Figure 4.8 provides information about the accuracy of the model in understanding the type of entity received on each user query. In both X and Y axis are documented all different supported entities. These entities are responsible as described in section 4.2.2.2 to extract specific information from user messages and are used by the chatbot during conversation handling so to produce targeted responses. The understanding of the correct entity each time is vital to the correct dialogue flow of the chatbot. The Figure 4.8 shows that most entities are well separated from each other not creating many ambiguities. The ideal case would be that all entity matches would be correct and so in this confusion matrix all values to be on the diagonal. While most of the entities seem to be well separated there is one, the "adware_id" that seems to create many ambiguities with half of the times being mistakenly identified by the model as not entity at all. That is a problem that may affect the dialogues supporting "adware_id" and should definitely be resolved as a future work.

The proposed chatbot interprets the entities using regular expressions trying to match the expected format of the entity. These regular expressions are structured in a way to be generalised in order to match all possible entities may be provided (e.g. spyware names) from a user. These regular expressions have a structure like

[A-Za-z0-9.]+, which means that it a expects an entity having possibly letters, numbers or even full stop punctuation for word separation.

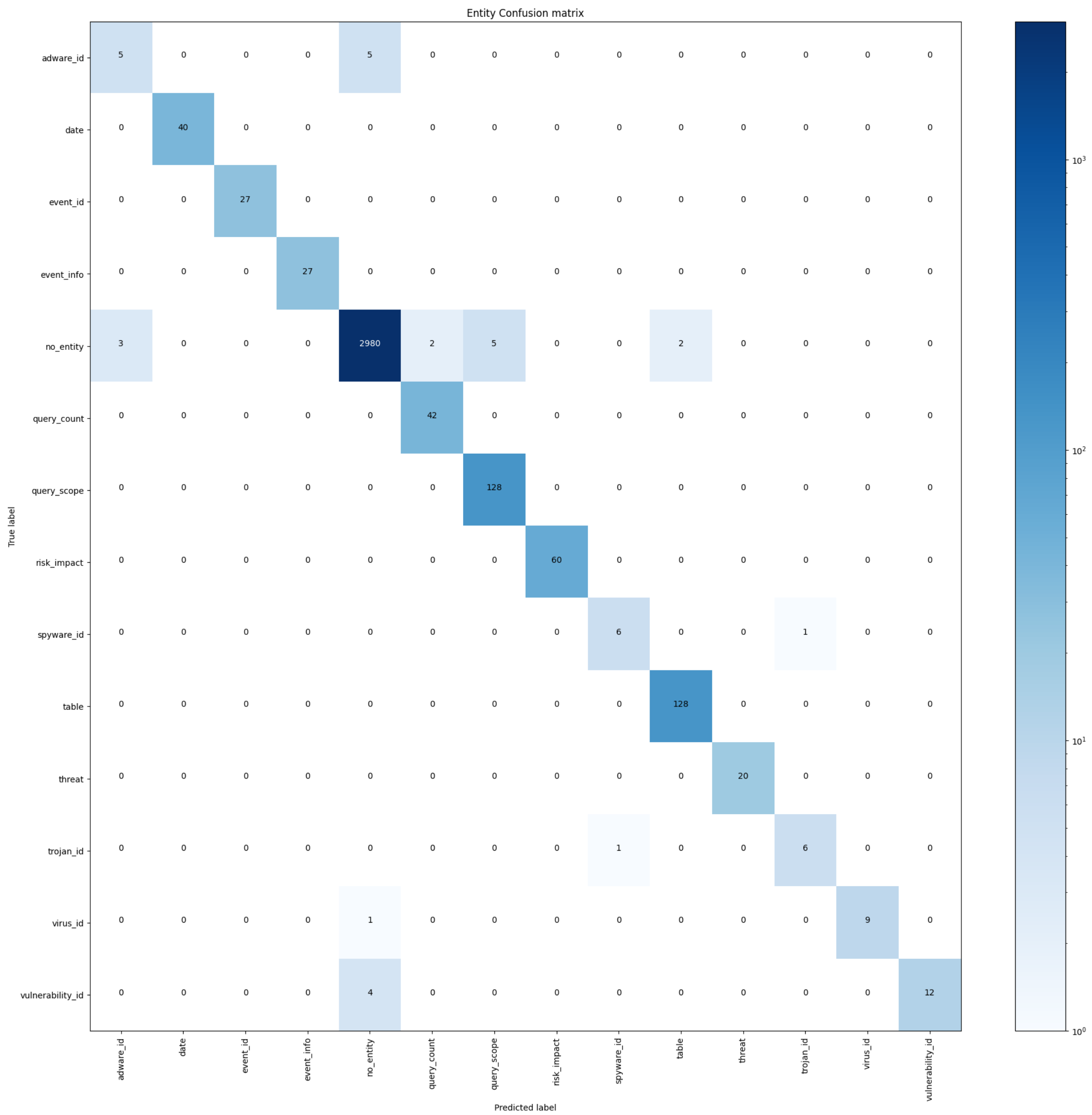


Figure 4.8: DIET Classifier Confusion Matrix - Entities

In the Figure 4.9 following, is represented the confidence of the model while predicting an entity in both correct and wrong situations. In the X axis are the number of the samples while on the Y axis is the percentage of confidence. In the correct part of the figure, the correct predictions of most entities have happened with a confidence around 98%. On the other hand, on the part of the figure showing the wrong predictions of entities, only a fraction of the samples are found compared with the correct predictions. Also the confidence of these wrong predictions vary from 52% to 98%. These confidences provide an assurance, combined with Figure 4.8, that most entities will be correctly discovered and with a great confidence level.

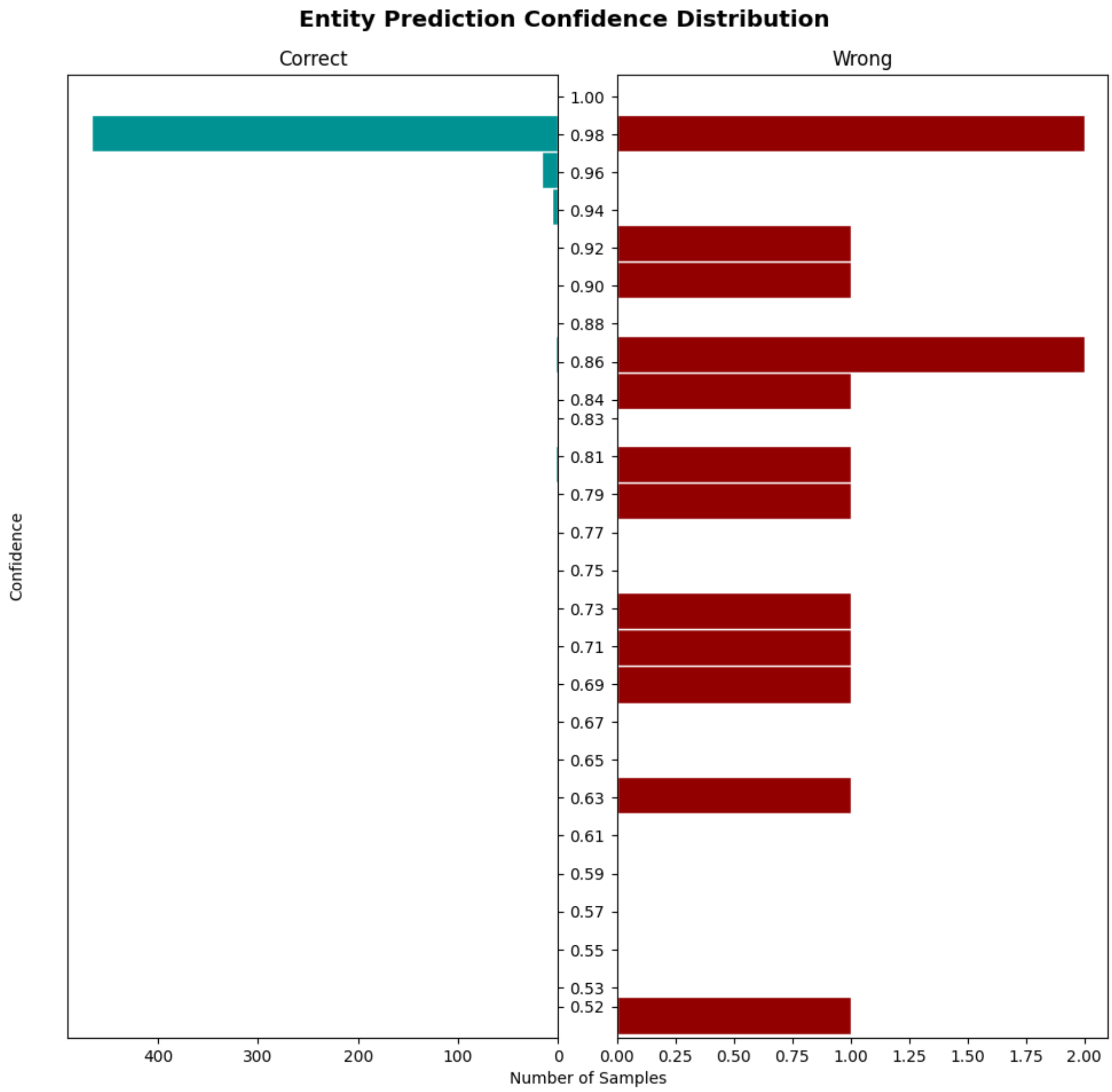


Figure 4.9: Entity Prediction Confidence Histogram

The Figure 4.10, shows the intent prediction confidence distribution. Similarly, as in Figure 4.9, the plot has two parts for correct and wrong predictions. As the figure suggests, intent predictions were correctly identified for most of the samples (more than 600) while the majority of them are with confidence of 97%. On the other hand, on the part of the wrongly predicted intents, the distribution of the wrong predictions is shown. These few samples show a wide distribution of confidences from 25% to 97% with most of them around 97%. The absolute number of these wrongly predicted cases is a small fraction of the correct ones but is very significant to understand why this happen and which intents leave space for ambiguity.

Eliminating the wrong predictions of intents was a long procedure throughout the creation of this proposed chatbot. Responsible for that is the NLU.yml file where the intents are declared and the corresponding examples are documented. By populating each intent with a plethora of example queries, the predictions were made more robust in ambiguities. But the most crucial step on the differentiation of intents was the step of differentiating the entities found in each query. Thus, the good intent predictions are based and strongly connected to the predictions of entities as described in the Figure 4.9.

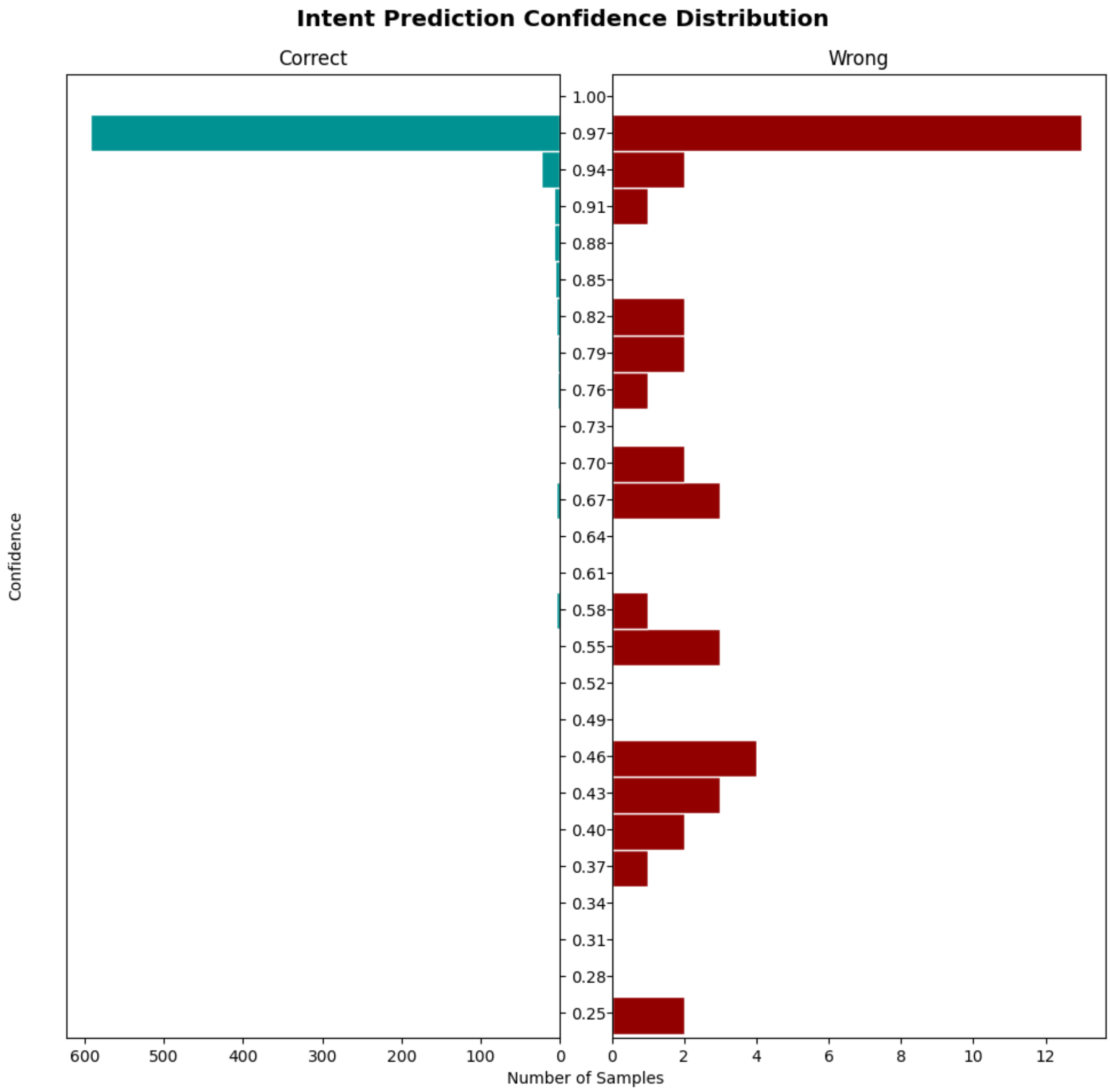


Figure 4.10: Intent Prediction Confidence Distribution Histogram

Throughout the implementation of this chatbot, these metrics were constantly checked in order to find the balance between well separated intents and entities.

Overall the Chatbot's training results are promising and indicate a well-trained and balanced chatbot. After **300 training epochs** the model achieved:

- a relatively low **total loss = 1.06** meaning that the learning mechanism from the training data worked very well.

- The **intent accuracy = 0.991** is very high meaning that the chatbot is capable of correctly identifying user intents in almost every case.

- The **entity F1 score = 0.986** also high indicating that precision and recall are high also and well balanced. This metric shows that the chatbot can very accurately extract entities from the provided user messages.

The training process has been done numerous times attempting different number of epochs. With epochs less than 300 (e.g. 50, 100) the intent accuracy and F1 score were a bit lower but in experiments with more than 300 (e.g. 350, 400) epochs no significant change in the accuracies has been documented. So in order to balance accuracy and training time, 300 epochs were used in the model's training procedure.

These metrics indicate that the chatbot has been trained well and is likely to perform accurately in real-world user questions.

4.4.2 Comparison with General Purpose Chatbots

On that section, a comparison will be held between the chatbot proposed in this work and the famous and powerful Large Language Model-based chatbots Google's Gemini and OpenAI's ChatGPT.

To do so, the same question was held in both Gemini and ChatGPT to provide information about the trojan named Android.Geplook. The resulted outcomes are shown in Figure 4.11.

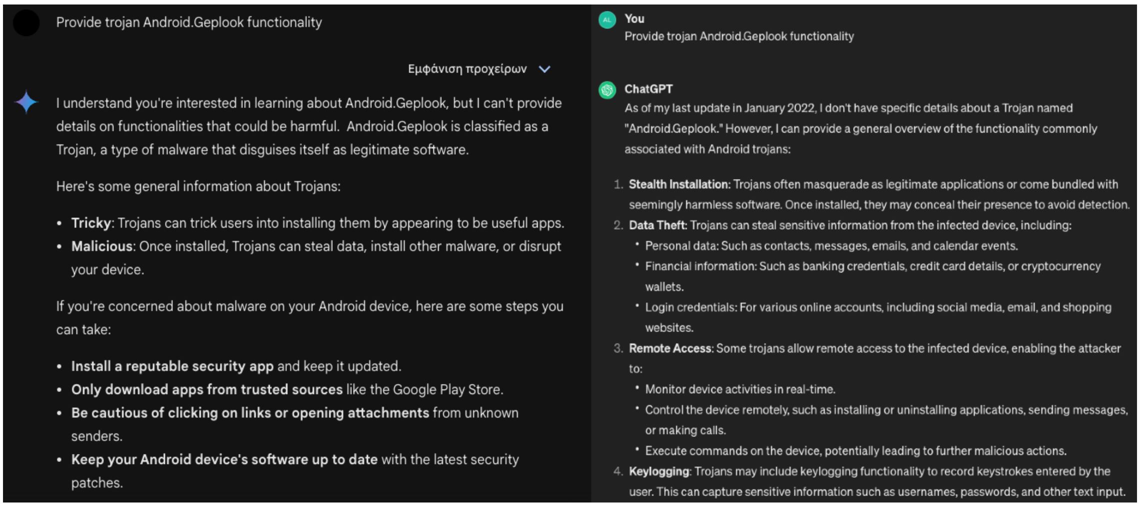


Figure 4.11: Gemini and ChatGPT on the same question

As observed, an examination in both Google's Gemini and OpenAI's ChatGPT reveals a limitation in their ability to address specific threats (e.g. trojan Android Geplook) and they cannot provide useful responses. Both these chatbots handle this situation by providing generic information about trojans and their characteristics without any information relevant to the entity of question. In contrast, in Figure 4.12, on the domain-specific chatbot proposed in this work, a user can have multiple information about this threat and even make more in-depth questions about it than just a description. This contrast indicates the significance of domain-specific chatbots despite the presence of powerful general-purpose Large Language Model chatbots.

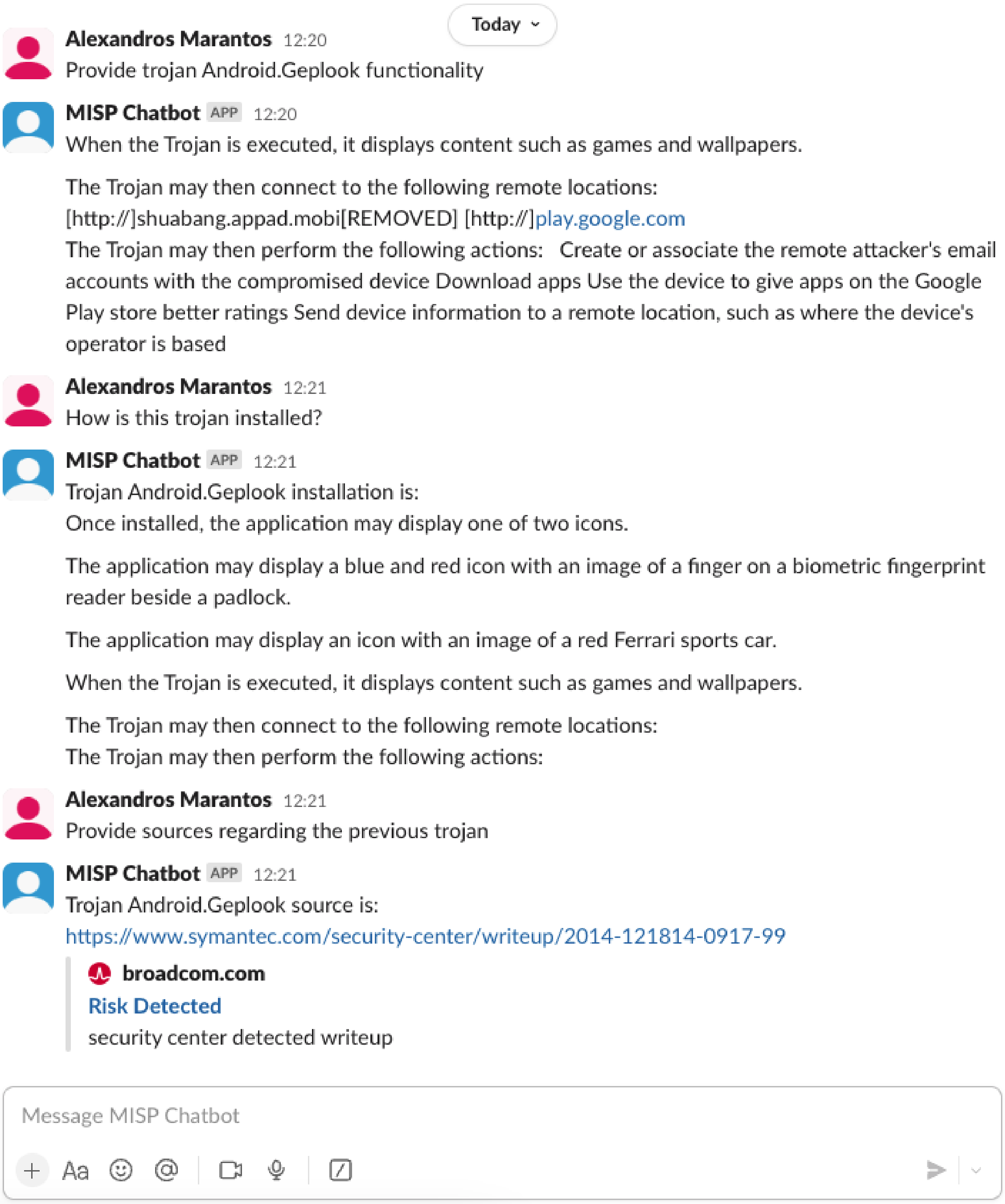


Figure 4.12: The proposed chatbot answering a question about a specific trojan

Chapter 5

Chatbot Deployment and User Interaction

5.1 Chatbot User Manual

This section describes the steps needed to launch and deploy the proposed chatbot. There are some prerequisites as described above and are mentioned again here:

- Downloaded codebase and database dumps on the local machine.
- Installed Rasa version 2 or higher and Python 3 programming language.
- Configuration of a publicly accessible URL via Ngrok for integration with Slack.

For the deployment process the steps needed follow:

- Two Linux terminal windows are required.
- In the first terminal, the actions.py file is executed using the command **rasa run actions**. This deploys the action server component of the chatbot. This terminal should be up and running throughout the duration of the deployment.
- In the second terminal, train the model based on the available data using the command **rasa train**.
- Last step, deploy the chatbot application using the command **rasa run**.

5.2 Chatbot Example Interactions

To make the user experience more immersive, the chatbot has the ability to engage in basic conversations like interactions with a human representative on a support chat window. This functionality, as shown in Figure 5.1, includes greeting, expressions of gratitude, questions regarding the chatbot itself and even asking general questions about MISP.

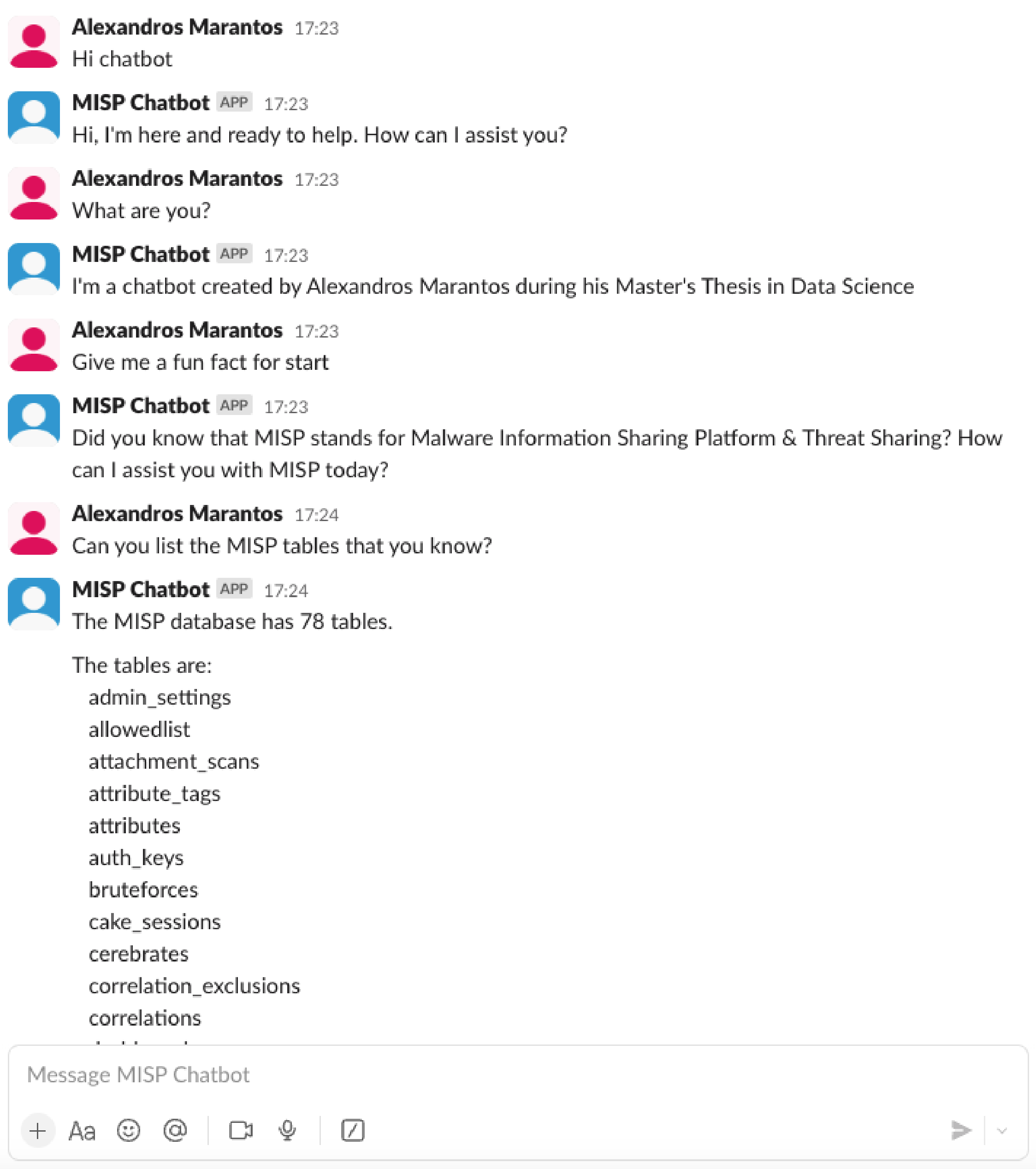


Figure 5.1: Simple dialogue

The following screen capture (Figure 5.2) shows an interaction where the user asks information about a specific MISP event by providing its name. To find the requested information, the chatbot searches within the MISP SQL database. This query, as listed below, returns the value of the description for the user-specified event.

SELECT value1 FROM attributes WHERE event_id = (SELECT event_id FROM attributes WHERE value1 = event name) AND object_relation = 'description'

After the name of the event was received from the user, the chatbot can manage to save it as an entity within the current session by implementing the Entity Recognition algorithms described before. This enables the chatbot to have a memory of the user's previous event of interest. So, when later on the conversation the user asks about the date of *this* event, the chatbot can recognize the reference to the previously specified event. That way the chatbot has all information needed and starts the process to query the database and get the desired response. Also, the same way the user can ask other questions such as references or sources that have more information about this event. The chatbot returns a list of all the urls, providing information about it.

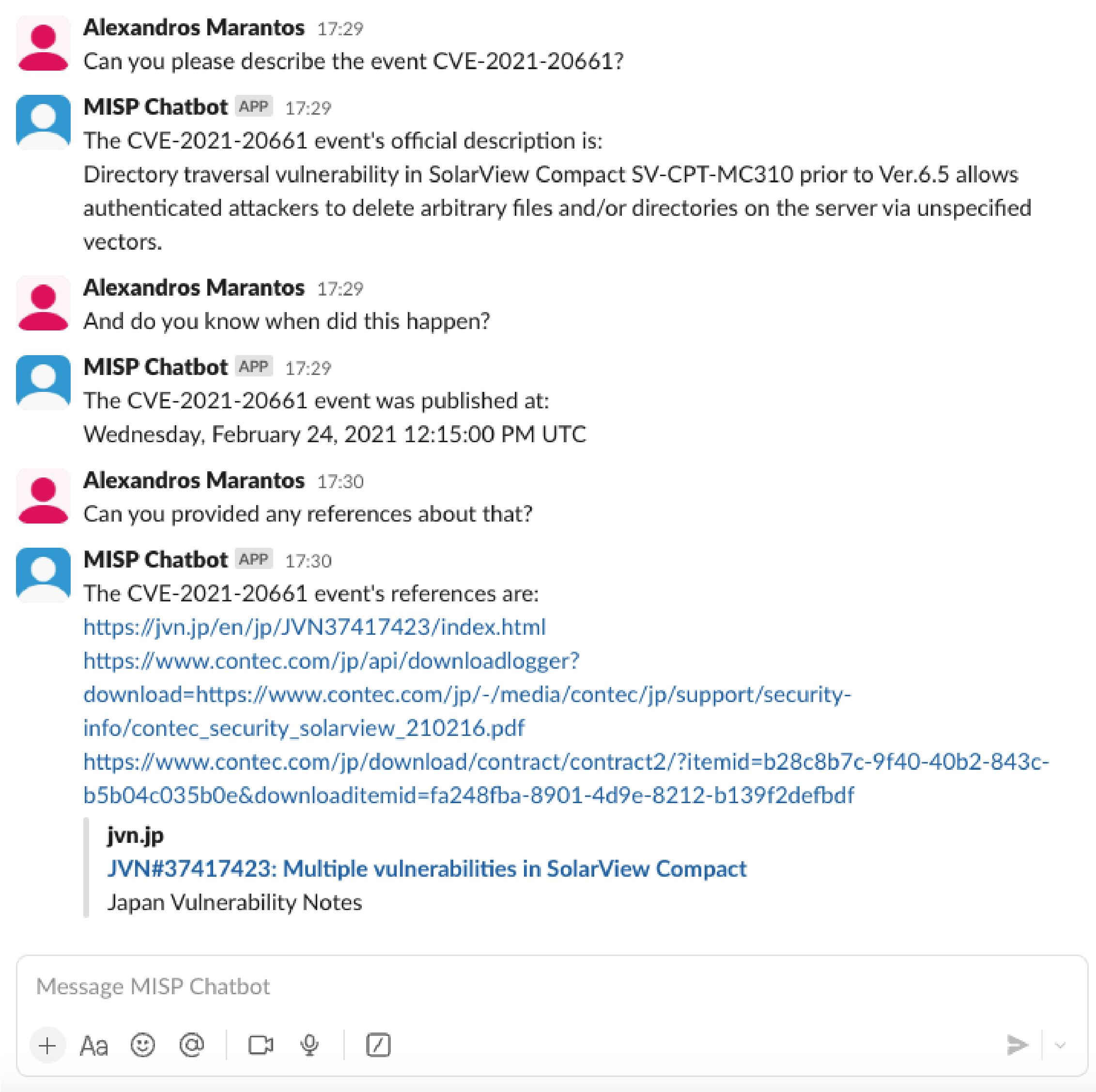


Figure 5.2: An example conversation regarding the about MISP event

The conversation example in Figure 5.3 showcases the user asking information about the different threat levels in vulnerabilities and threats. After getting this information, the user asks a question to get the amount of threats specified with low severity. The chatbot returns a numerical value representing the total number of threats found identified by the specified risk impact. The chatbot provides additional information to that inquiry in an attempt to make the user experience better, it categorizes the low severity threats and presents a list of all available vulnerability categories. For each category a count of the corresponding number of threats with low risk impact is represented next to its name. Naturally, this is the case if the question was about medium or high risk threats.

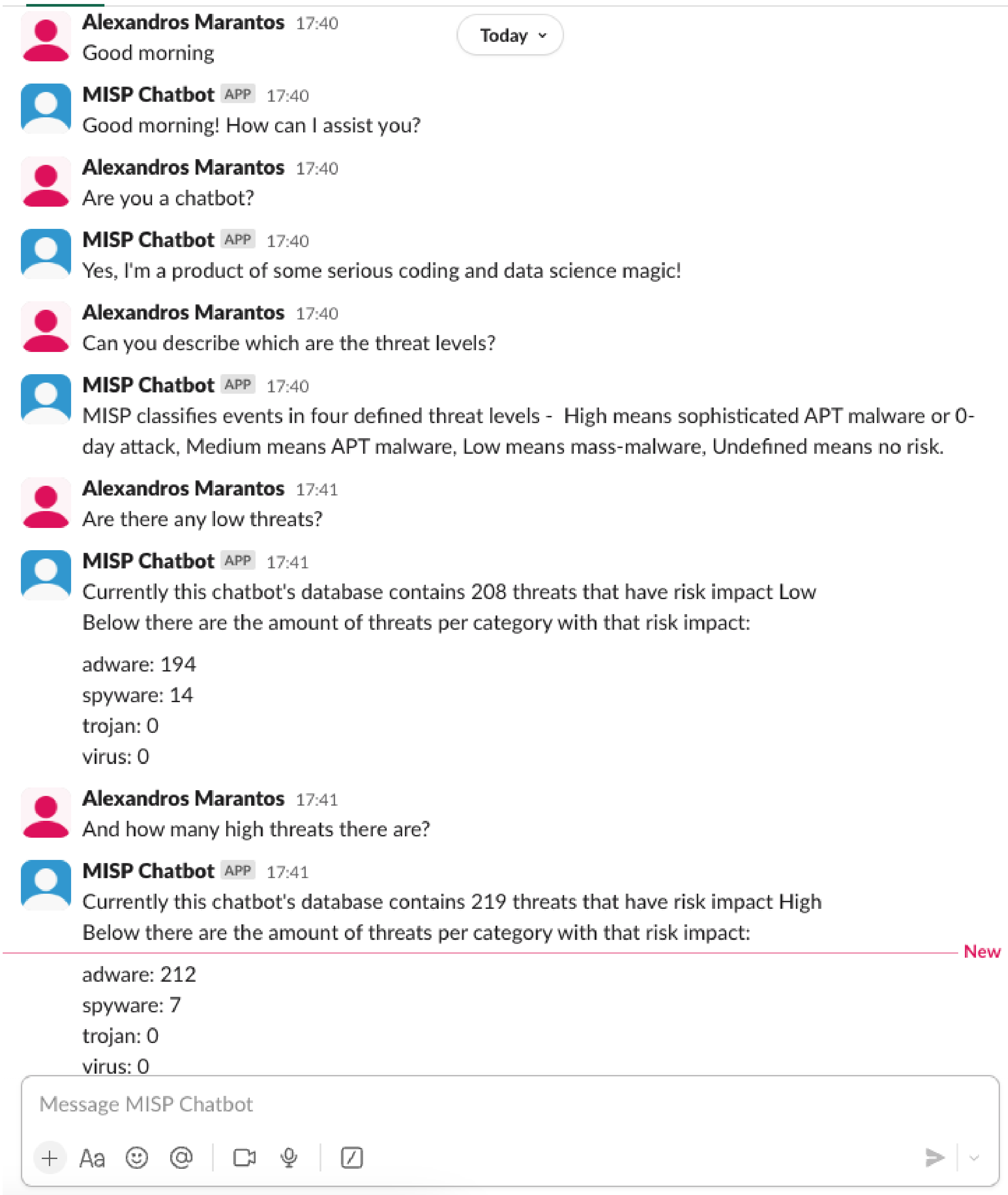


Figure 5.3: An example conversation, about threats' risk impact/level

The following dialogue in Figure 5.4 presents the capability of the chabot to provide information about specific threats or vulnerabilities by their name, querying through JSON files this time. This functionality refers to threats categorized as adwares, spywares, viruses, and trojans but can be extended if other JSON files, including different threat types, are added. After retrieving the initial information about a particular threat, the user can ask more questions related to that without the need to specify again its name. The chatbot is able to remember multiple entities within different categories and establish conversations about various threats simultaneously.

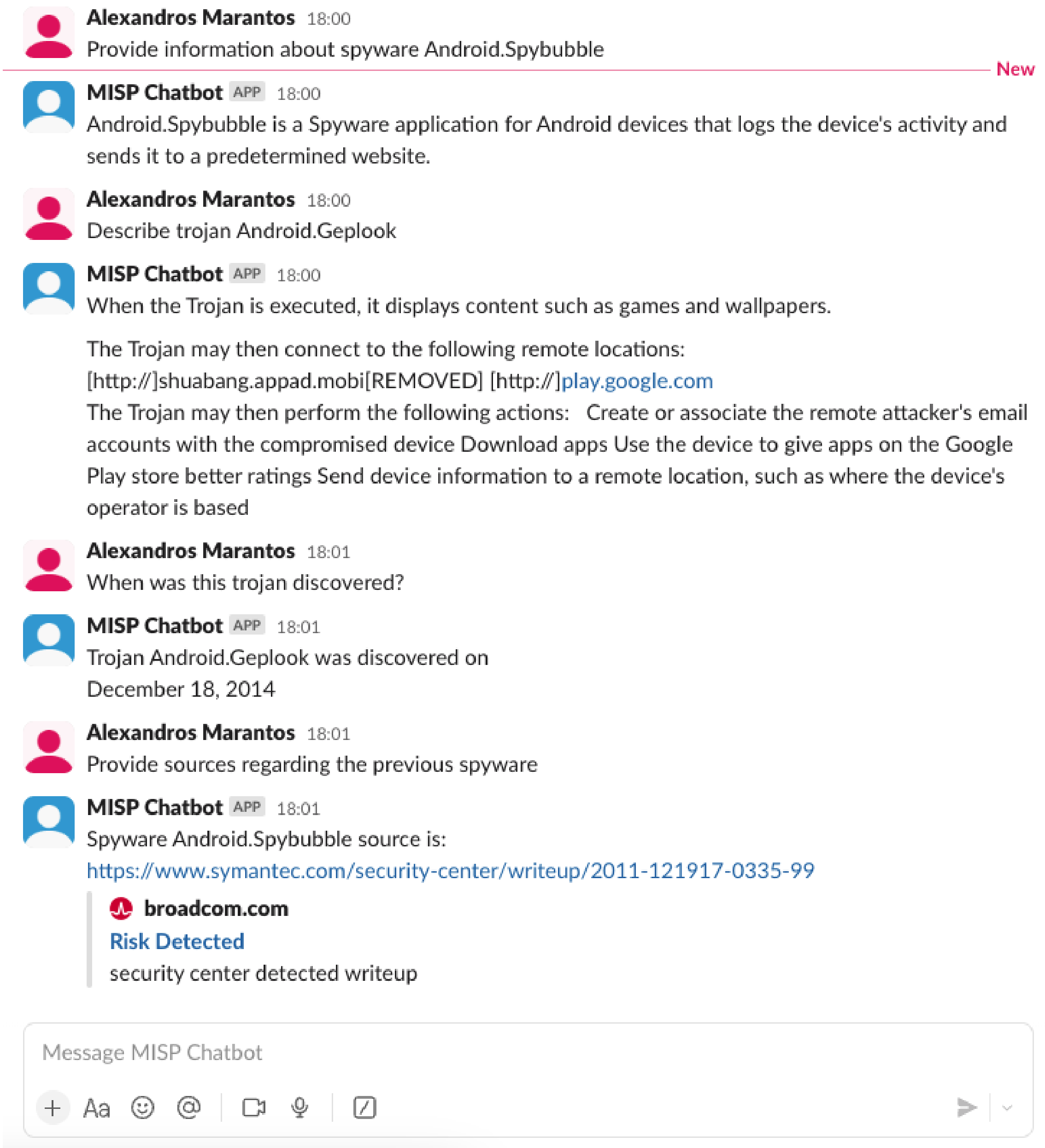


Figure 5.4: An example conversation, about vulnerabilities general information

The following conversation, presented in Figure 5.5, provides an illustration of how a user can learn about a particular threat by making a series of different questions. The supported questions cover all meaningful metadata a threat has. The example follows an adware as an example but similar conversations can be held for all supported threats.

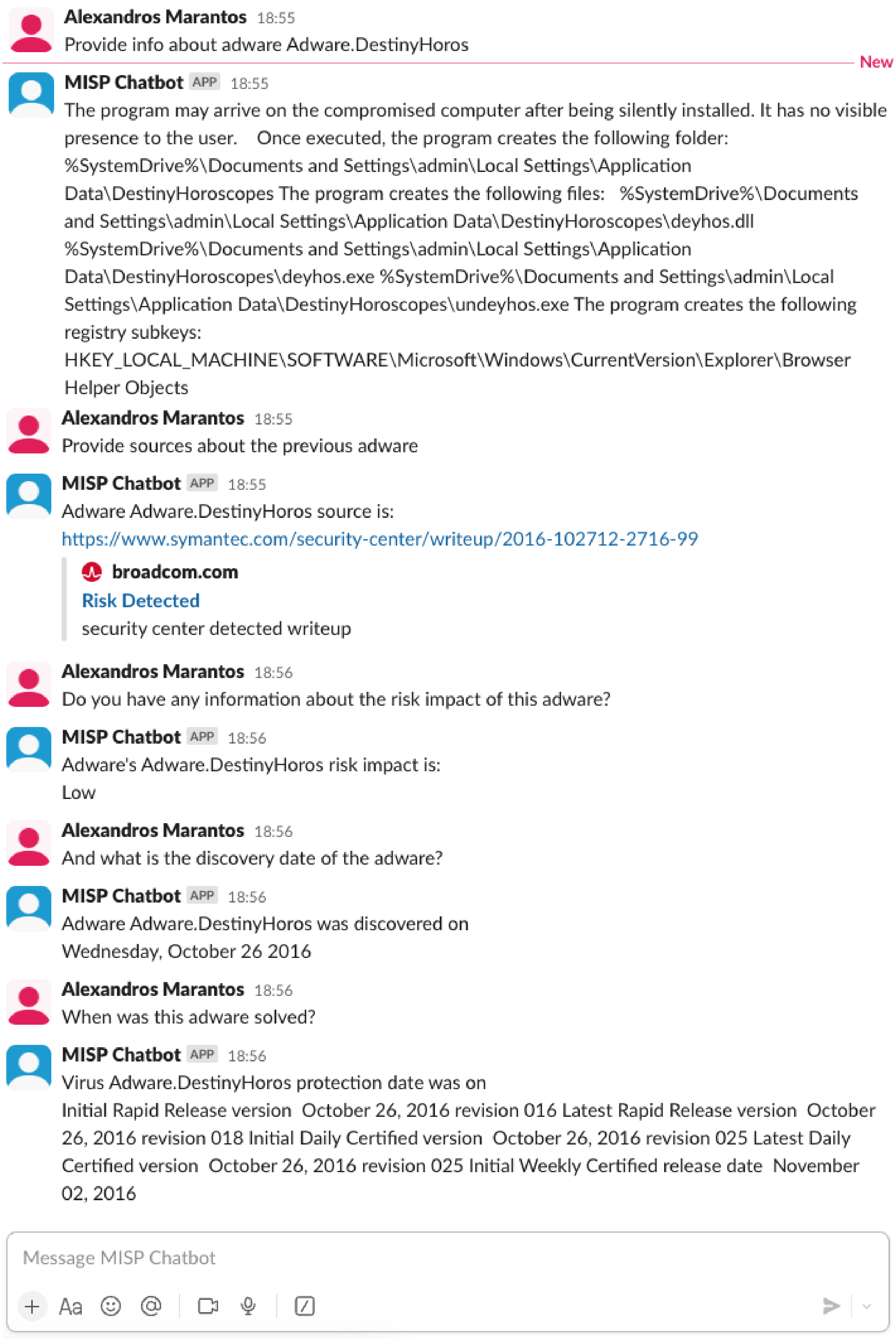


Figure 5.5: An example conversation, where the user asks for adware general information

The chatbot can also handle questions that include the name of a threat but ignore the type of it. In this case, in Figure 5.6, the chatbot provides the functionality of finding the type of this threat. After that procedure, the user can continue regularly with other questions regarding it. The chatbot has memorized both type and name and can search accordingly in the database.

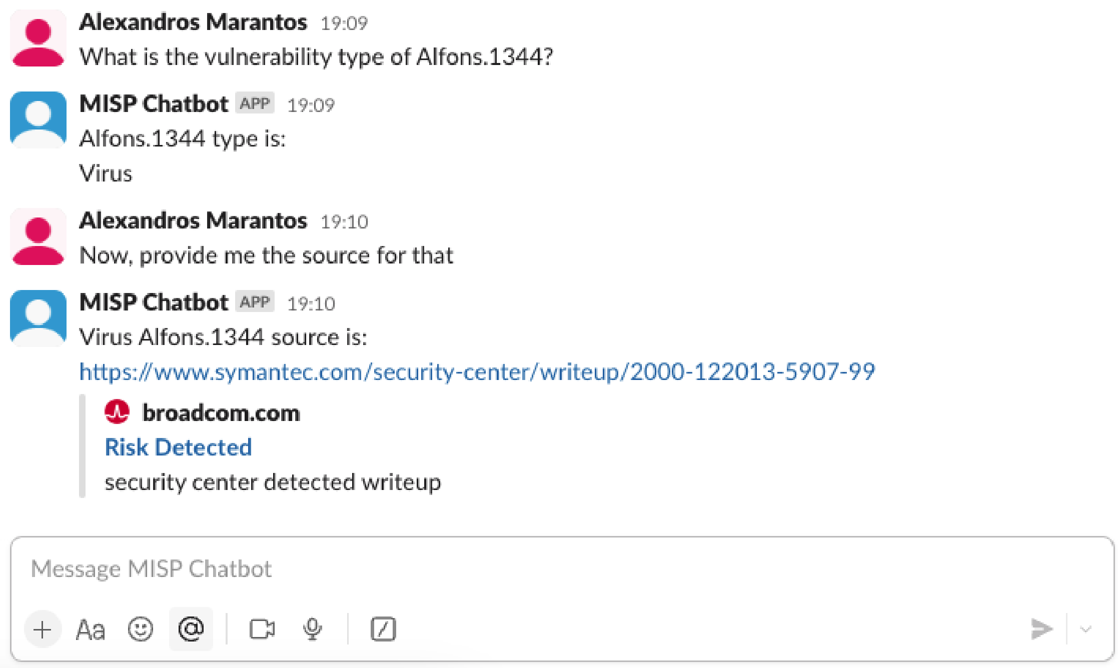


Figure 5.6: An example conversation, where the user asks for a vulnerability with unknown type

It is important to note that the proposed chatbot offers a valuable statistical information. It provides responses containing information about the number of threats existing either before or after a particular date (see for example Figure 5.7). Also, as was described in this section, the response incorporates a categorized breakdown. That way the chatbot represents the exact number of threats belonging to various threat categories that were discovered before or after a user-specified date.

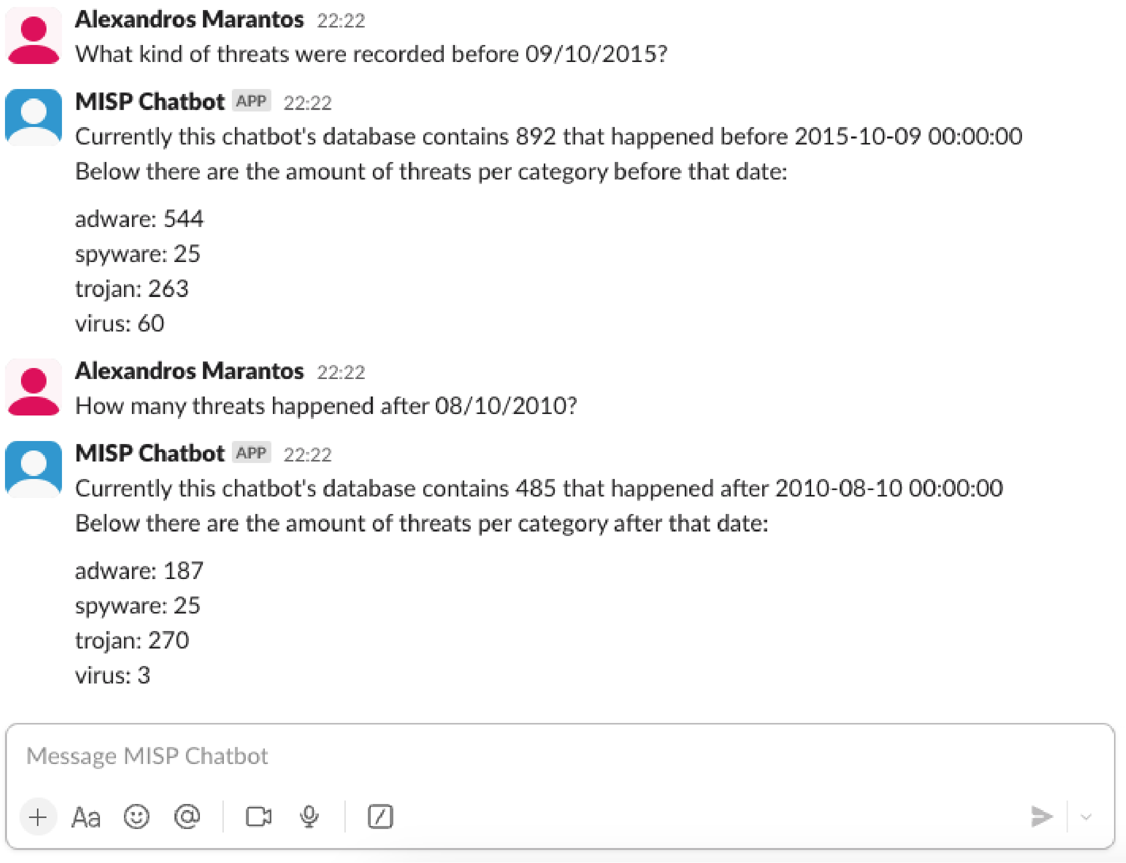


Figure 5.7: Dialogue about the number of threats before/after a date

The previous searching capability can be even more extended by adding as an extra limit the risk impact factor.

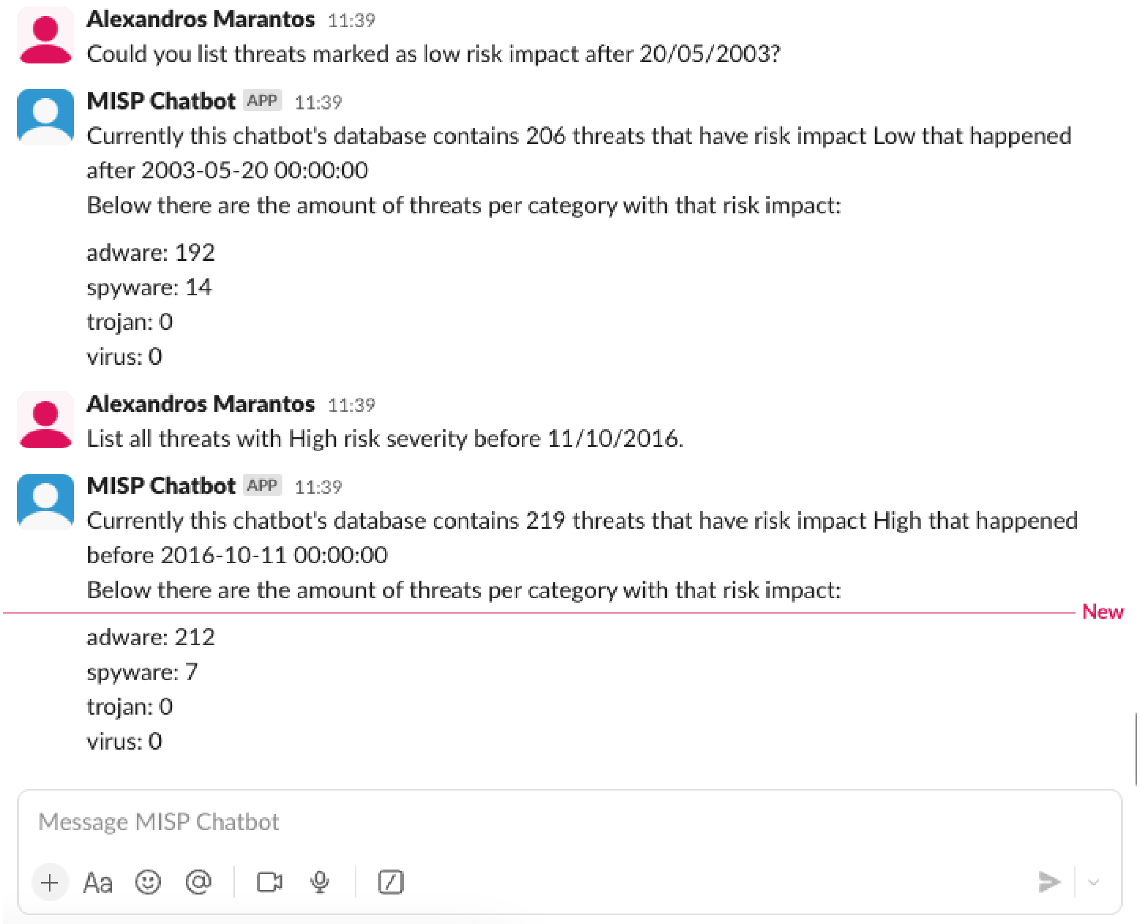


Figure 5.8: Dialogue about counting threats by risk impact and before/after a date

The chatbot finally, supports commands like "*give me the latest threat with low risk*". This question despite a simple one, can provide a lot of information. First, by checking the latest threat, either with stating its severity or not, a user can find out if the chatbot's database is updated or when was the last time it did. Another aspect and more crucial is that, that way a user can have an easy insight about the fields a threat might include and also which of the chatbot supports.

The included screenshot also shows the chatbot deployed within the Slack application on a smartphone so to illustrate the user interface and overall appearance when accessed from a mobile device.

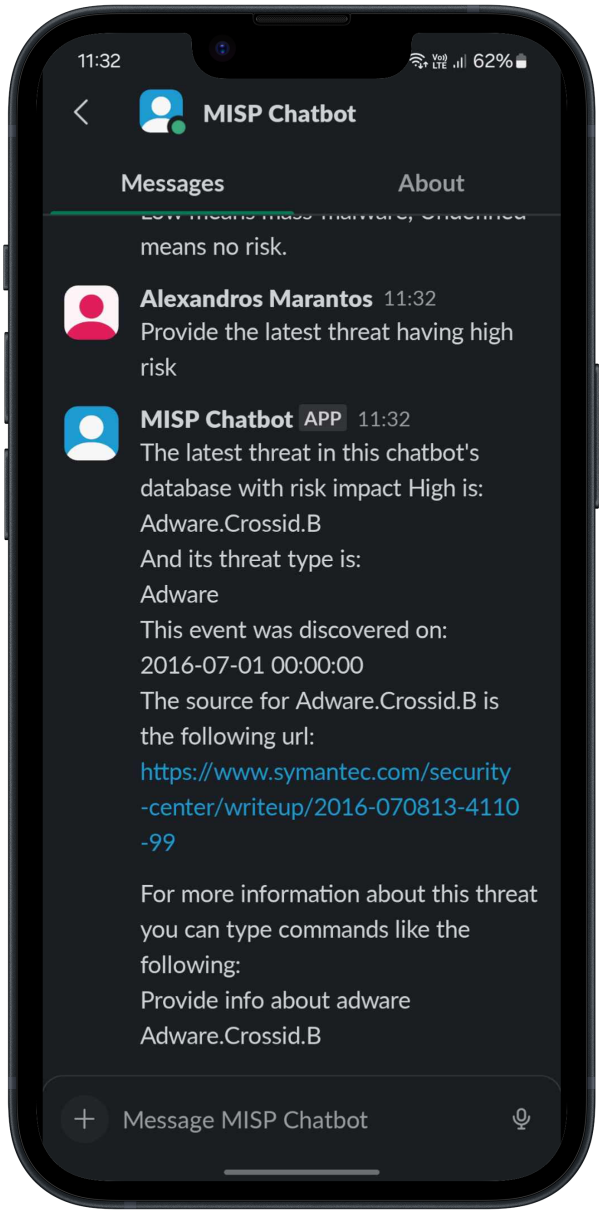


Figure 5.9: The user asks the chatbot about getting the latest threat by risk impact

Chapter 6

Conclusions and Future Work

6.1 Conclusion

This dissertation has explored the development of a cybersecurity chatbot designed to connect users and the enormous amount of information available within different cybersecurity databases. As the cybersecurity threats explode in the recent years, efficient tools are important for exploring this field. Chatbots provide user-friendly interactions and automate information retrieval.

The existence of numerous cybersecurity databases provide significant information. However, exploring these resources can be a very inefficient process and useful information may be missed. This project attempts to propose a way to address this challenge by proposing a chatbot doing exactly this. This chatbot interacts with various kinds of databases, is open-source and is effortlessly expandable. This chatbot serves information about cyber threats in a presentable and easily reachable manner, leading to hopefully an improved experience.

This thesis has explored the specific details of the developed chatbot. Chapter 2 provided a literature review of existing on cybersecurity and other domain-specific chatbots, analyzing their technological foundations. Chapter 3 offered a detailed exploration of the integrated databases and their data structures. Chapter 4 presented the architecture and underlying technologies implemented through the chatbot's development. At last, Chapter 5 provided a presentation of the dialogue functionalities and the information extraction capabilities of the chatbot as a user would encounter them.

In conclusion, this thesis has presented the design and implementation of a cybersecurity chatbot that attempts to make it easier for the users to explore the vast cybersecurity information stored in various databases. As the field of artificial intelligence grows at the current rapid pace the capabilities of chatbots like the one presented here will grow even further. Chatbots like this are presented on peer-reviewed papers more and more every year on many different domains. This project attempts to be part of this exciting era.

6.2 Future Work

While this project has established a foundation for an innovative cybersecurity chatbot, it is recognized that is a step towards a truly valuable tool in the cybersecurity domain. To become even more valuable some further development should be done as due to time restrictions it couldn't be done during that thesis. So as next steps, to make this chatbot a well established tool some additional aspects would be introduced. The first one is the integration of Large Language Models within the chatbot and the second is a web scraper that updates the databases on real time.

Introducing a Large Language Model (LLM) into the chatbot's architecture would significantly elevate its capabilities on the natural language processing. This technology would allow for more natural interactions. Users could pose complex questions and receive responses that are not only accurate but also tailored to the specific phrasing and context of their inquiries. Rasa offers various libraries for LLM integration so developing this is a feasible and very promising work.

Currently, the chatbot relies on databases found and downloaded over the internet through cybersecurity databases. Implementing a web scraping feature would add into the chatbot the capability to retrieve data dynamically from these, and possibly more, sources in real-time. This feature would guarantee that the chatbot has access to up-to-date threat information. This feature would benefit a lot the tool as users want something always up-to-date to search into. Providing the latest insights would also strengthen its overall effectiveness in awareness around cyber threats.

In conclusion implementing these improvements will help the chatbot become a promising tool in the cybersecurity field. This tool will provide users with up-to-date knowledge and a natural user experience.

References

[1] Definition of chatbot in english by lexico dictionaries. *https://www.lexico.com/en/definition/chatbot.*

[2] Definition of chatbot in english by dictionary.com. *https://www.dictionary.com/browse/chatbot.*

[3] Attia Qammar, Hongmei Wang, Jianguo Ding, Abdenacer Naouri, Mahmoud Daneshmand, and Huansheng Ning. Chatbots to chatgpt in a cybersecurity space: Evolution, vulnerabilities, attacks, challenges, and future recommendations. *CoRR*, abs/2306.09255, 2023.

[4] Joseph Weizenbaum. ELIZA - A computer program for the study of natural language communication between man and machine (reprint). *Commun. ACM*, 26(1):23–28, 1983.

[5] Kenneth Mark Colby, Franklin Dennis Hilf, Sylvia Weber, and Helena C. Kraemer. Turing-like indistinguishability tests for the calidation of a computer simulation of paranoid processes. *Artif. Intell.*, 3(1-3):199–221, 1972.

[6] Dr. sbaitso — wikipedia. *https://en.wikipedia.org/wiki/Dr.$_s$baitso.*

[7] Veton Këpuska and Gamal Bohouta. Next-generation of virtual personal assistants (microsoft cortana, apple siri, amazon alexa and google home). pages 99–103, 2018.

[8] In 2016, Microsoft's racist chatbot revealed the dangers of online conversation. *https://spectrum.ieee.org/in-2016-microsofts-racist-chatbot-revealed-the-dangers-of-online-conversation.*

[9] Gpt-3 powers the next generation of apps. *https://openai.com/blog/gpt-3-apps.*

[10] Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. Is ChatGPT good at search? investigating large language models as re-ranking agents. pages 14918–14937, December 2023.

[11] Gpt-4 is openai's most advanced system, producing safer and more useful responses. *https://openai.com/gpt-4*.

[12] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, and Anja Hauth et al. Gemini: A family of highly capable multimodal models. 2023.

[13] Abid Mehmood, Arslan Shafique, Moatsum Alawida, and Abdul Nasir Khan. Advances and vulnerabilities in modern cryptographic techniques: A comprehensive survey on cybersecurity in the domain of machine/deep learning and quantum techniques. *IEEE Access*, 12:27530–27555, 2024.

[14] Malware: Viruses, spyware, adware other malicious software. *https://www.umass.edu/it/security/malware-viruses-spyware-adware-other-malicious-software*.

[15] Chatgpt sets record for fastest-growing user base. *https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/*.

[16] Shafquat Hussain, Omid Ameri Sianaki, and Nedal Ababneh. A survey on conversational agents/chatbots classification and design techniques. 927:946–956, 2019.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[18] Yoon Kim. Convolutional neural networks for sentence classification, 2014.

[19] L. C. Jain and L. R. Medsker. *Recurrent Neural Networks: Design and Applications*. CRC Press, Inc., USA, 1st edition, 1999.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[21] Ammarah Farooq, Muhammad Awais, Sara Ahmed, and Josef Kittler. Global interaction modelling in vision transformer via super tokens, 2021.

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[23] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[24] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Bing Yin, and Xia Hu. Harnessing the power of llms in practice: A survey on chatgpt and beyond, 2023.

[25] Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers, 2021.

[26] Lei Zhang, Yuge Zhang, Kan Ren, Dongsheng Li, and Yuqing Yang. Mlcopilot: Unleashing the power of large language models in solving machine learning tasks, 2024.

[27] Muriel Figueredo Franco, Bruno Rodrigues, Eder John Scheid, Arthur Jacobs, Christian Killer, Lisandro Zambenedetti Granville, and Burkhard Stiller. Secbot: a business-driven conversational agent for cybersecurity planning and management. pages 1–7, 2020.

[28] Tanja Bunk, Daksh Varshneya, Vladimir Vlasov, and Alan Nichol. DIET: lightweight language understanding for dialogue systems. *CoRR*, abs/2004.09936, 2020.

[29] Rasa Technologies. Rasa: Open Source Conversational AI. *https://rasa.com*.

[30] Sofian Hamad and Taoufik Yeferny. A chatbot for information security, 2020.

[31] Prasasthy Balasubramanian, Justin Seby, and Panos Kostakos. Transformer-based llms in cybersecurity: An in-depth study on log anomaly detection and

conversational defense mechanisms. In *2023 IEEE International Conference on Big Data (BigData)*, pages 3590–3599, 2023.

[32] Aigerim Mansurova, Aliya Nugumanova, and Zhansaya Makhambetova. Development of a question answering chatbot for blockchain domain. *Scientific Journal of Astana IT University*, 15(15):27–40, Sep. 2023.

[33] Pragnya Ramjee, Bhuvan Sachdeva, Satvik Golechha, Shreyas Kulkarni, Geeta Fulari, Kaushik Murali, and Mohit Jain. Cataractbot: An llm-powered expert-in-the-loop chatbot for cataract patients, 2024.

[34] Cynthia Wagner, Alexandre Dulaunoy, Gérard Wagener, and Andras Iklody. Misp -the design and implementation of a collaborative threat intelligence sharing platform. 10 2016.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[36] Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.

[37] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[38] Ledell Wu, Adam Fisch, Sumit Chopra, Keith Adams, Antoine Bordes, and Jason Weston. Starspace: Embed all the things!, 2017.

[39] Rasa Components. Rasa: Open Source Conversational AI. *https://rasa.com/docs/rasa/components.*

[40] Rasa Whitespace Tokenizer. Rasa: Whitespace tokenizer. $https://rasa.com/docs/rasa/reference/rasa/nlu/tokenizers/whitespace_tokenizer/.$

[41] Tanja Bunk, Daksh Varshneya, Vladimir Vlasov, and Alan Nichol. Diet: Lightweight language understanding for dialogue systems, 2020.