University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Development and Refinement of a Chatbot for Cybersecurity Support

Bulin Shaqiri
Zurich, Switzerland
Student ID: 17-701-442

Supervisor: Muriel Franco, Eder Scheid
Date of Submission: February 4, 2021

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Cybersecurity is increasingly getting the attention it deserves. At the latest now that the Covid-19 pandemic has gripped the entire world and employees are forced to work from home wherever possible, thereby creating new vulnerabilities for cybercriminals to exploit, the importance of cybersecurity is even more evident.
In the past years, however, the investments as such have been rising, with the vast majority moving towards security as a service, and thus contracting off-site protection from various providers. Together with recommender systems, the sheer volume of solution alternatives can be managed, but still requires expertise to correctly specify the requirements. End-users are therefore not enabled to enter their demands in a simple and quick way. While other fields have explored conversational agents (i.e., chatbots) as possible solutions, including some approaches in cybersecurity, there has still been no work that has used such conversational agents to improve cybersecurity management. In this sense, the overall objective of this thesis is to provide a prototype that allows end-users to submit their requests for cybersecurity support, with the conversational agent then responding with accurate answers, so that the insightful information extracted from the conversation can be used by end-users during the cybersecurity decision-making process.

Acknowledgments

First, I would like to express my deepest gratitude to my supervisor, Muriel Franco, for his unwavering support, interesting discussions, and valuable and insightful advice and suggestions throughout the thesis. It has been such a pleasure and incredibly inspiring to work with someone as enthusiastic and passionate as Muriel.

I would like to extend my sincere thanks to Prof. Dr. Burkhard Stiller, my co-supervisor Eder Scheid, and everyone else from the Communication Systems Group (CSG) involved in the SecBot project, for giving me the opportunity to work on such a fascinating topic.

Finally, I also wish to thank my family and friends for their endless support and encouragement all throughout this time.

Contents

Chapter 1

Introduction

Cybersecurity emerges to be one of the key player of the digital era. With the ever-increasing enhancement of information technology and the resulting connectivity throughout the networks, not only opportunities for businesses, but also new threats within the operating cyberspace arise.
One such threat concerns the increasing number and complexity of attacks commited by cybercriminals. Every 39 seconds, a cyberattack is conducted, leading to an average of 2'244 attacks daily. Alone in the first six months of 2019, more than 4 billion data records were compromised due to data breaches [57]. Moreover, during this Covid-19 pandemic, in which the corona virus forced employees to work remotely from home wherever possible, the number of cyberattacks is not expected to decrease.
People with IT-background are usually aware of such cyberthreats and know already basic terms and concepts of cybersecurity whereas people without such a background often do not even know about the basics which should already belong to common knowledge. This could lead to serious consequences. [2], for example, highlights that only 61% of the participants, that took part of the respective survey, could choose the correct definition of a phishing attack from multiple provided answers. Also, 45% confessed to use the same passwords for different applications, and one of the most worrying result is that 90% admitted to using devices provided by their companies for private purposes [2]. Such behaviour clearly does not correspond to the cybersecurity hygiene recommended of today.

There already exists a variety of security measures such as the standard ones including firewall, virus scanner or validation systems. Then there are also other, more advanced measures like sandboxing, bug bounty programs or even extensive security audits. And since employees are nowadays the weakest link in the chain of security policy in an organization's cyberspace, employee training has become absolutely vital [35, 44].
At this point, it must also be mentioned that not all companies actually have the same capacity to spend on cybersecurity. Large enterprises such as banks or insurance companies naturally have a greater budget to spend on such protection compared to small and medium sized enterprises (SME) where to some extent around 37% of employees lack basic digital skills. In addition, SMEs often don't see cybersecurity as a major part of their digital strategy and therefore do not feel any need resp. sometimes do not have the necessary ressources to compete with larger companies for new security talents. This

somehow affects also larger businesses in a negative way. Cybercriminals have identified SMEs as easier targets and try to enter the supply chain of larger businesses through compromised systems of smaller ones [21]. Verizon pointed out that in 43% of security breaches in 2019, small businesses were the victim [1].

1.1 Motivation

Nevertheless, companies have indeed been investing in cybersecurity protection throughout the years with the spendings on such cybersecurity tools to be expected to reach $123 billion in 2020 [13]. No less than 50% of these investments relate to security as a service, which is based on cloud-delivered security through the use of various technologies [27]. This model offers businesses numerous benefits. Always using the newest and most up-to-date security tools as well as having experienced security professionals working for you are just few of them [6]. However, with a pool of thousands of suppliers, it is likely that the solutions offered may differ in terms of the vendors focus, such as supported cyberattacks, costs, performance or simply the technology being used [4].
Being spoilt for choice, it becomes difficult for customer to get a clear overview of the different possible solutions and choose the best one based on their requirements. In addition, not having a security specialist on board makes the decision-making process even more challenging, especially when you are under attack. Thus, it is crucial to know which off-site services come up with appropriate shielding. For that, software tools, which are specialised in retrieving and filtering information in order to make best possible recommendations, are a possible solution. Such tools are also known as recommender systems and are widely used in many different areas [55].
In this regard, [25] presents a cybersecurity management support tool which is capable of recommending appropriate protection services according to different demands such as region, deployment time, and price conditions. Nevertheless, this recommender system does not enable end-users to specify their requests in an easy and efficient manner. In order to define a precise cybersecurity strategy, meaningful interactions between the end-users and specialist systems have to be provided.
In this sense, other areas have also been exploring and came up with an artificial intelligence tool called chatbot as a viable solution. Chatbots are designed in a way that they are able to have a conversation with a human in natural language and therefore are able to facilitate the human-machine interaction by using various input methods [59].
However, there is no work exploring chatbots to improve cybersecurity management. For [25], for example, chatbots could be used to allow end-users to specify their demands in natural language, thereby helping to refine filters that determine which protection services best meet the user's needs.

1.2 Description of Work

The main goal of this bachelor thesis is for the student to implement new features and knowledge into the prototype of the chatbot called SecBot, which has been initially devel-

oped by the Communication Systems Group (CSG) of the University of Zurich. Besides development of new features and knowlegde, the thesis also demands the refinement of the inital implementation of SecBot. Hence, by finishing the thesis, SecBot should be able to give appropriate answers regarding the requested cybersecurity support, so that users can take those insightful responses into account for their cybersecurity decision-making process.
In order to accomplish this goal, the student first needs to carry out basic research in the areas of cybersecurity and chatbots. Once the basic understanding is acquired, the student has to investigate the actual SecBot implementation and concurrently understand Rasa, the actual framework being used, including the concepts of Machine Learning and Natural Language Processing. After the acquirement of the technical background the student then has to get familiar with the main requirements of cybersecurity management. This step enables the student to provide SecBot with new information and custom actions. Afterwards, the student will have to train the chatbot and will therefore provide new stories that cover the new scenarios and information flows. In a concluding chapter, the new features and improvements shall be evaluated with use cases and then discussed.

1.3 Thesis Outline

Chapter 2 provides a common knowledge which serves as a foundation to understand the new features and enhancements.

Chapter 3 explores related work with an eye set on conversational chatbots.

Chapter 4 introduces SecBot in more detail. It gives a thorough overview of SecBot's capabilities and how it leverages current chatbot development concepts. It also discusses the initial implementation before classifying SecBot using the metrics presented in Chapter 2.

Chapter 5 first explicitly defines the scope of the work and provides a non-technical overview of the approach. This is then followed by a high-level description of the new features that provide enhancements or new capabilities to the prototype.

Chapter 6 provides a more technical view of the implementation of SecBot. Thus, implementation specifics of the configuration, the attack description feature and the attack support feature are presented in a detailed manner. In addition, instructions are given on how to integrate SecBot with the popular messaging application Telegram.

Chapter 7 first evaluates the implemented features based on a case study and then proceeds with a discussion of SecBot's performance, limitations, and challenges.

Chapter 8 summarizes the work on this thesis and finally concludes this project.

Chapter 2

Background

This chapter provides you with the general knowledge and concepts that serve as a basis to understand the improvements and novelties presented in the thesis. First, I will introduce some main threats that have become common in today's cyberspace. Up next, the concept of natural language processing will be explained before presenting the notion of chatbots. Finally, a brief introduction to the framework used, Rasa, will be given.

2.1 Cybersecurity Threats

Cybersecurity threats have always been a big issue and still continue to be so today. These include various cyberattacks which are becoming increasingly more sophisticated and difficult to defend against. From the wide variety of different cyberattacks, I have selected three attacks that are presented in more detail in the subsequent sections.

2.1.1 Distributed Denial-of-Service (DDoS)

A distributed denial-of-service (DDoS) attack aims to disrupt or even take out online services and their resources. It is similar to a denial-of-service (DoS) attack with the only difference being how the attack is carried out. DDoS attacks are decentralised, which means that the attack traffic does not come from one single source as it is the case in DoS attacks. Instead, the cybercriminals make use of multiple compromised systems, also known as botnets, with each infected device becoming a bot [11, 16]. This not only allows to intensify the attacks, but also helps to cover the cybercriminal's traces on the net [28]. Upon instruction, the hacker-controlled botnet then starts sending huge number of requests to the victim's server simultaneously, consuming a significant amount of its resources. The server has then no longer the capacity to react to legitimate traffic, which leads to a denial of service [11].
DDoS attacks can be decomposed into three large categories: volumetric, protocol-based and application-layer based DDoS attacks, with each attack targeting a different layer of

the network connection. Table 2.1 outlines all key characteristics of the different types of DDoS attacks, which are also presented in more detail in the following sections.

Table 2.1: Types of Distributed-Denial-of-Service attacks

DDoS			
Type	Characteristics	Scope	Challenge
Volumetric Attacks	Generation of substantial traffic volumes to saturate the bandwidth and produce traffic congestion.	Focus on bandwidth depletion.	Mitigation of threats that combine reflection/amplification attacks with botnets.
Protocol-based Attacks	Over-consumption of server resources and network intermediaries (e.g. firewalls, load balancer, etc).	Focus on resource depletion.	Weaknesses of Internet protocols (e.g. TCP, UDP, ICMP, etc.).
Application Layer Attacks	Exploit weaknesses in the application layer in order to disrupt resp. take out applications, online services or websites.	Require less amount of bandwidth than the other attacks while still being at least as disruptive.	The challenge lies in distinguishing legitimate traffic from malicious.

Volumetric Attacks: Volume-based DDoS attacks, also known as bandwidth consumption attack, appear to be the most common category. Cybercriminals usually utilize botnets to create sheer volume of traffic to literally flood a website, which then becomes overloaded, thus causing network congestion. As a consequence, legitimate users are prevented from interacting with the website. Volumetric attacks are measured in bits per seconds (bps).[54].

- **DNS-Amplification:** During a DNS-Amplification attack, the cybercriminal makes use of the weaknesses of the domain name system (DNS) servers. The perpetrator sends small queries in which the IP addresses have been changed to the target's one, to open DNS resolver, which in turn reply with larger DNS responses to the spoofed IP address of the victim. Instructing every bot within a botnet sending similar request, the sheer volume of traffic heading towards the victim's server grows even more significantly [31].

Protocol-based Attacks: In contrast to volumetric attacks, protocol-based DDoS attacks aim to use up both server resources as well as network appliance resources. They are therefore also rightly referred to as state-exhaustion attacks. These attacks make use of the vulnerabilities within the network and transport layer (OSI Layer 3 and 4), such as forging protocol requests, to deny access to the victim's service. The scale of measurement is expressed in packets per seconds (pps) [11, 54].

- **SYN-Flood:** A SYN-Flood attack takes advantages of weaknesses in TCP handshake. To establish a TCP network connection the client sends first a SYN packet, which are basically initial request packets, to the target server. The server in turn replies with a SYN/ACK packet to confirm the communication request. This reply must then also be confirmed by the client by sending the ACK packet to the target server. During a SYN-Flood attack, the cybercriminals block the third part of the handshake process. Waiting for the ultimate confirmation of the handshake, which obviously will never arrive, the victim's server remains with many open connections, thus being rendered unavailable by exhausting its resources. [10].

Application-layer Attacks: Also referred to as layer 7 DDoS attacks, the aim of the application layer attacks is to fabricate a denial-of-service of an application, online service, or a website by consuming a great amount of the victim's server and network resources [11]. Contrary to volumetric and protocol-based attacks, these attacks require much less resources. The network requests themselves are not spoofed and therefore appear to be legit at first sight (e.g. TCP connection). This makes it extremely tough to differentiate the malicious traffic from the legitimate [54].
What makes these kind of attacks even more threatening is the varying necessity of resource usage during a client-server interaction. A simple request from the client side can, for instance, enforce running large database queries on the server side. This asymmetric nature of some Internet protocols can overload the server during an attack, thus rendering it inaccessible to legitimate users [8]. The power of such an attack is expressed in request per seconds (rps) [54].

- **HTTP-Flood:** Cybercriminals typically use botnets to overload the target's server with ressource-intesive HTTP requests, with the goal of exhausting the server's finite resources such that it is no longer capable of responding to legitimate traffic. Moreover, mitigation of such attacks becomes more complex as it is difficult to distinguish these requests from legitimate ones since they do not use spoofing or reflection techniques. There exist different HTTP-Flood variants [9, 41] .

2.1.2 Malware

Malicious software, also known as malware, denotes a generic term that covers all software programs which deliberately cause damage to systems or other software programs. There exist a broad range of attacking threats trying to sneak into your system, reaching either to your data or core system functionality to derange the ordinary operations [37, 12].
From this pool of possible attacks, I will discuss three of them in more detail in the following sections and set out the main findings in Table 2.2.

Table 2.2: Types of Malware

Malware			
Type	Characteristics	Scope	Challenge
Virus	Self-replication into other programs to either exploit vulnerabilities in the compromised system or cause damage to it.	Requires a host program which must be actively executed by a user.	Detection and mitigation of evolutionary or new viruses.
Worm	Self-replication into other systems, targeting entire networks to create large botnets and resp. or damaging the infected ones.	Stand-alone program which does not need host files or user interaction.	The ability to spread quickly and very easily over a network.
Ransomware	Blocks users from accessing their system or personal files and requests payment of a fee to regain access.	Requires other cyberattacks to spread.	Ransoms are hard to track, as allmost all payments are made in cryptocurrency.

Virus: A virus is a malicious program which is capable of reproducing itself into other executable programs or documents. To achieve this, the virus initially needs a host program, which must then be actively executed by a user to be able to propagate further. These infected executables are then in turn able to spread these malicious bits of codes to other executable programs or even exchange them completely with a duplicate of itself [36]. Viruses can be hugely damaging to your system. Apart from infecting other programs, the core functionalities, applications or existing data of the compromised system may be manipulated or deleted [14]. With millions of viruses lurking around the Internet, it becomes almost impossible to get all their definitions and signatures into a database so that virus detection programs could identify and remove them, especially when it comes to evolutionary resp. newly designed viruses [34].

Worm: Computer worms are yet another type of malicious software. They closely resemble viruses and are therefore often incorrectly used as synonyms.
Worms make also use of self-replication to reproduce themselves and infect other not yet compromised systems, where they then stay active for as long as possible [51]. Other than viruses, worms do neither need a host program nor active user interaction. They are self-contained programs that strive for full access to the target system. Once they find a way into your system, the duplication process starts and distribution takes places via networks and internet connections. Since all duplicates inherit the same abilities, all systems that were initially connected to an infected one are put at risk, with the insufficiently protected systems becoming contaminated [30]. In this way, cybercriminals attempt to infiltrate complete computer networks with the aim to establish a botnet with

which further cyberattacks can be conducted [58]. Moreover, worms also entail so-called payloads. A payload is basically a malicious attachment containing other malware such as ransomware or viruses, which in turn may attack and harm the imposed system in their particular way. Besides, worms are also able to set up back doors to enable other malicious software to exploit the compromised system at a later time [30].

Ransomware: Ransomware presents another form of malware, where cybercriminals particularly pursue monetary incentives. There are many variations of ransomware with screen-lockers and encrypting ransomware being the two main types [52].

- **Screen-lockers:** As the name implies, screen-lockers, or in short also only called lockers, do block the access to your computer. Your data is not stolen or damaged. You only see a screen including the information that you have been locked out and payment instructions that need to be followed in order to regain the access [52]. Cybercriminals often combine these attacks with pretending to be law enforcement authorities, such as the FBI, who freezes you out of your computer with the claim that there has been illegal activity, and then charge you a fine [38].
- **Encryption ransomware:** Encryption ransomware, on the other hand, encrypt critical data and files with complex algorithms such that no software or system is able to decrypt them. Even tough you pay the demanded ransom, it is no guarantee that the attacker provides the necessary private key to restore the access to the files [52].

Ransomware is becoming more and more popular, especially due to payments made in cryptocurrency, where cybercriminals traces can't be tracked [12]. It has gone even to the extent where you now can purchase ransomware as a service (RaaS) in the darknet to cheaply carry out such attacks [40]. By 2021, Cybersecurity Ventures expects such ransom attacks to be conducted on businesses every 11 seconds, skyrocketing the cost of damages up to $20 Billion USD [60].

2.1.3 Phishing

The third cybersecurity threat I want to introduce is a cyberattack which relies on the concept of social engineering. During a phishing attack, the cybercriminals' objective is to induce their targets to reveal precious and personal information such as (company) login credentials, credit card information or business data. To achieve this, phishers often spoof links or websites or even exploit vulnerabilities of genuine websites where they do hidden and malicious redirects [46]. Moreover, phishing attacks also misuse human psychology. After gaining the victim's trust, cybercriminals often exert pressure by exploiting the victim's emotions (e.g. exploiting the fear of reputational damage) or generate curiosity with the attachment [62]. Such phishing scams either end with cybercriminals financially enriching themselves with the stolen information, or the attack served to deploy further malware for future major cyberattacks [7].
In the following sections, I will present three main types of phishing and summarize the key points in Table 2.3.

Table 2.3: Types of Phishing

Phishing			
Type	Characteristics	Scope	Challenge
Standard Phishing	Attempt to entice targets to disclose sensitive and valuable data.	Targeting a large number of users and expecting that only a few will be victimized. Requires only minimal preparatory work from the criminal.	Encouraging users to take steps and protect themselves. Apart from that, phishing techniques are constantly being innovated, which makes them difficult to detect.
Spear Phishing		Focused attacks on organizations or specific individuals. Additional information must be collected from the criminal in advance.	
Whaling		Focused attacks on senior or C-level management. Requires maximal preparatory work from the criminal.	

Standard Phishing: This subform of phishing emphasizes quantity over quality. In this sense, phishers focus on affecting a broad number of individuals, assuming that only a few of them will be victimized. Using forged communications channels, such as emails, cybercriminals craft them to look legitimate, thereby fooling their targets into disclosing sensitive data through fake links and websites. Consequently, this approach does not require attackers to prepare much work in advance [7, 46].

Spear Phishing: Contrary to standard phishing, spear phishing attacks are specifically tailored to companies or particular individuals. To lunch such attacks, cybercriminals need to do further investigations on the target's side and then enrich the emails with the collected information, such as personal data or data about the organization, to let the message appear more trustworthy, and thus tempt them to open malicious links or attachments [46, 47].

Whaling: Whaling is a special form of spear phishing, where the upper management of organizations are set as targets. This includes the senior level management such as CEO, CIO, CFO and other high-level professionals. What makes them attractive is the fact that they have access to business-critical data (e.g. intellectual assets, customer information, etc.) which could be traded for big money on the black markets. As this target group is extremely well-trained in phishing attacks, phishers must do extensive research beforehand and whale at the most opportune time [46, 48].

Indeed, phishing attacks are very popular. In 2019, 32% of all data breaches were related to such attacks [1]. Since cybersecurity is improving, so do phishing attacks, which are constantly being refined to fool more people or get more outcome per victim [46]. Other types that have emerged over time are voice phishing (also known as vishing), clone phishing or sms phishing (also known as smishing).

2.1.4 Countermeasures

Tackling a DDoS attack is difficult in terms of discriminating between genuine and malicious network traffic. A possible approach is using a web application firewall (WAF) that is able to stop malicious requests. This is done with the help of rules, which are designed to detect DDoS tools. User-defined rules can be implemented immediately to instantly react to such attacks. Otherwise, there is also the possibility of blackhole routing, where all the traffic is redirected to a null route and consequently discarded from the network. This measure suits best when a website that is part of a larger network is attacked, thus helping the rest of the network not to suffer from the immediate attack [11].

One obvious way to prevent malware is to install a quality anti-virus respectively anti-malware program which periodically scans your device. This program along with all the other installed software needs to be up-to-date in order to reduce the risk of a malware attack. For website administrators, website security scans have the same effects and help identifying and mitigating these kind of threats [12].
Moreover, backing up all available data on a regular basis has become inevitable. This way, in the event of a ransomware attack, for instance, you can immediately access the backed-up data and thus reduce the costs incurred for this attack [52].

When it comes to combating phishing attacks, the most effective countermeasure is to create awareness of these kind of attacks among all employees. In this sense, not only must security awareness training be conducted on a regular basis, but employees must also be trained through simulations so that they acquire the ability to identify phishing scams in real world situations [62].
Other useful measures include filters, which protects the user against spam, suspected links and attachments, as well as fostering a strong credential hygiene, where passwords are changed regularly with a certain complexity and authentication solutions are implemented [46].

2.2 Natural Language Processing

Natural Language Processing (NLP) is a subarea of artificial intelligence (AI) which is concerned with the human-computer interaction through the use of natural language. When transforming raw user input in the form of unstructured data into a structured form that computers understand, NLP uses various machine learning (ML) techniques to deduce meaning. In this sense, several NLP tasks help to decompose the input data [26]. One important component of NLP is Natural Language Understanding, or shortly NLU.

Characterized through the syntactic and semantic analysis of data, NLU tries to understand the meaning of the complete utterance. With the help of syntactic analysis techniques, NLU evaluates whether the natural language conforms to grammatical rules. This is done by using various methods such as lemmatization, part-of-speech tagging or stemming. Then techniques of semantic analysis are used to understand and interpret both the words and the structure of the sentences. In this regard, algorithms such as the named entity recognition (NER), which identifies words or expressions as valuable entities, or word sense disambiguation, which assigns a contextual meaning to a word, or the natural language generation, which converts structured data into natural language, are used [18, 26, 33].

2.3 Chatbots

A chatbot is a software program which is based on artificial intelligence technologies. Also known under various other names, such as conversational agents, AI chatbot, digital or virtual assistants, chatbots do enhance the human-computer interaction in a pleasant way by simulating natural language conversations with its human end users. They can be integrated into a variety of technologies allowing such conversations to take place on websites, messaging applications, or on mobile apps [59, 19].

Nowadays, chatbots are present in almost every industry. The companies benefit above all from efficiency at the operational level and cost savings, especially in the area of customer service. Non-stop availability, lower customer latency, scalability or improved customer retention are just some of the aspects that contribute to these cost savings, which are expected to be around $8 billion by 2022, according to Juniper Research [43, 50]. No wonder chatbots increase in popularity.
There exists thousands of chatbots with varying degrees of intelligence, which can be classified into two main types:

- **Transactional Chatbots:** Also referred to as task-oriented or declarative chatbots, transactional chatbots aim towards accomplishment of certain tasks. They guide the user in a very well-structured and specific manner and often best improve the user experience through automation of processes, trying to eliminate the user's reliance on experts or additional interfaces. If needed, these chatbots also take over the interaction with other external systems. Furthermore, thanks to NLP, transactional chatbots also exhibit basic AI skills that make them capable of processing and answering simple queries or transactions [64, 42].
- **Conversational Chatbots:** Steered through data and predictions, conversational agents are more ambitious than transactional chatbots. They engage users in interactive, dialogue-oriented conversations where the chatbots may store important information for later purposes. To achieve this, the conversational agents make use of various AI technologies, including NLP, NLU and machine learning algorithms. This ensures that they are always fully conscious of the context in which the conversation is taking place. Moreover, these chatbots can also be personalized so that they can learn about the interests of users and make appropriate suggestions [42].

Based on my investigations I created the following taxonomy to furhter divide conversational chatbots into three subforms:

Table 2.4: Subtypes of Conversational Chatbots

Conversational Chatbots		
Type	Approach	Limitations
Rule-based Chatbots	Function like a decision-tree where predefined rules generate responses (question-answer pairs).	Only show basic capabilities where questions outside of the predefined set of rules can't be answered. Might become extremely labor-intensive to develope.
Machine Learning Chatbots	More dialog-oriented, data-driven and predictive. They require sample data sets to be able to generalize from the training processes. They also learn from patterns and previous experiences.	Such chatbots are like black boxes. If something goes wrong with the model, it is difficult to intervene, optimize or improve. There is also a lack of training data and other regulations for protection of the collected data.
Hybrid Model	Combination of rule-based and machine learning chatbots.	Black box and data protection issues remain.

Also known as linguistic-based model, rule-based chatbots anticipate what customer might want to know and define answers to those likely questions. These question-answer pairs are specified as rules and vary in the degree of complexity. They follow an if-then flow which makes the conversation highly specific and structured. Questions that lie outside the defined scope cannot be answered. Besides the slow development of such chatbots there are also some benefits, such as faster training phase or easy integration with existing systems [3].
This model can be further refined. For this purpose I present the following two categories:

- **Menu/button-based Chatbots:** This is an example of a chatbot with no required AI skills. Based on a predefined knowledge database the chatbot offers the user certain alternatives to click on and follow a particular scenario. Questions won't be given any attention as users can only click themselves through the decision tree. Consequently, this type of chatbot is very straightforward in its nature and is slowest when it comes to providing the desired response [45].

- **Keyword recognition-based Chatbots:** These kind of chatbots use NLP to extract keywords from the users' queries and match them with the correct answer using

AI. When there are many similar questions, such chatbots tend to have problems disinguishing between them and possibly give wrong answers [45].

Machine learning chatbots, or AI chatbots, make use of AI technologies and NLP to extract the right intentions from the user's utterances within the relevant context and then respond in natural language [3]. Contrary to rule-based chatbots, AI chatbots can understand queries other than defined within the training data. They are developed in such a way that over time they are able to learn on their own from previous conversations and become more intelligent. This is achieved through different algorithms and patterns that are constantly fed with new data, as the chatbots experience a new situation with each individual user, thus also gaining greater decision-making capabilities [17]. Moreover, some AI chatbots use additionally sentiment analysis to provide users a unique experience. The use of machine learning chatbots offers clearly more advantages than disadvantages. However, one disadvantage worth mentioning is the fact that such chatbots work like black boxes. If there are any problems with the model, it is extremely difficult to react to them. Other drawbacks are the limited set of available training data which have to be prepared and processed by humans or regulations protecting the collected data [59].

The third subtype introduced is a hybrid model which combines the best features of both, rule-based and machine learning chatbots. By using this approach, businesses become faster and more flexible. If they don't have any data available, the provision of such sample data sets can be very expensive and take a significant amount of time. With the hybrid model, they can simply provide their rules in this case and then train the chatbot with real user input. On the other hand, when data is available, then using the features machine learning chatbots provide yield the best outcome [59]. In any case, this combination constitutes a win-win situation. Further, since the models in this combination complement each other, there are no limitations to be added. The black box and data protection regulation issues remain, but the original limitations of rule-based chatbots decreased considerably.

2.4 Rasa

Rasa is a conversational artificial intelligence framework [49] that assists in creating context conscious conversational agents by providing various machine learning tools. Since Rasa is open source software, it provides many highly customizable and powerful features that allow its users to influence the creation of chatbots to the fullest extent.
Being equipped with state-of-the-art NLP technologies including natural language understanding, Rasa easily handles unstructured user messages and converts them into a structured, machine-readable form such as intents and entities. Moreover, Rasa's dialogue management allows for conversations with varying degrees of complexity, such as off-topic switches or going back-and-forth. For this purpose, important contextual information can be persisted in bot-memories, which are also known as slots.
Another valuable feature is the possibility to train your chatbot interactively in addition to supervised learning [15]. In this sense, training data is directly created through the interaction with your conversational agent. There is also the possibility to give immediate

feedback and to correct wrong predictions. To wrap it up, the created chatbots can be integrated in various messaging channels such as Slack, Facebook, or Google Home.

Due to its open source nature, Rasa brings in some transparency into the black box problem, which was discussed in Chapter 2.3, since each algorithm can be looked up and adapted for its own purpose. No wonder that Rasa worldwide enjoys the trust of some big players like Adobe, UBS, BMW or HCA Healthcare.

Chapter 3

Related Work

In recent years, chatbots have experienced an extreme increase in popularity. Whilst in the beginning transactional chatbots were prefered, especially before crucial advances in machine learning and NLP, conversational agents are now being increasingly used. In the next few sections, some conversational chatbots from different areas will be presented and discussed.

For the field of e-business, [61] proposes a rule-based chatbot to support the customer service division. By using Artificial Intelligence Markup Language (AIML) templates to specify rules, the chatbot aims to automate the live chat support and thus be able to respond to raised questions and concerns.

In [32] the authors presents Lumi, a machine learning chatbot that allows intent-based networking (IBN) by using natural language. Using Lumi, network operators only have to input high-level policies in order to configure the network. Provider-specific languages or low-level configurations know-how are no more required. Moreover, after the relevant information and intents are extracted, Lumi requests confirmation from the operator and then checks for contradicting policies before the deployment. In this way, the chatbot can take feedback into account for learning throughout time.

With particular regard to security, [5] proposes a chatbot that provides employees without the necessary technical know-how with specialists security expertise about the company network without interrupting network security employees, who in turn can fully focus on the more complex security relevant tasks. For questions such as whether network traffic to a certain server is permitted, the chatbot would interact with required applications such as Network Security Management tools and would respond to the user in simple natural language.
In this sense, we can derive that in order to built this kind of converational agent, a machine learning chatbot approach must be chosen. Pursuing a rule-based approach would result in a highly labor-intensive development and at the same time bear the risk of missing on important rules.

The United Kingdom, on the other hand, presents an actual implementation of a security related conversational agent. The Cyber Helpline [29], a non-for-profit institution provides

UK citizens with a chatbot that offers immediate assistance in the event of cybercrime incidents for free. Victims can directly and at any time start the conversation with the chatbot, typing in freely their concerns in natural language. The chatbot then analyzes the input on its own with the help of NLP and machine learning (ML) and tries to correctly identify the attack and offering next few steps to combat the attack. In the case of more complex attacks, the chatbot prioritize and forward the incidents to human volunteers, which consist of cybersecurity experts and computer science graduates [29, 63]. Considering the taxonomy presented on Table 2.4, due to the use of various AI technologies and the ability to help without human interventions, we can further classify The Cyber Helpline's solution as a subform of machine learning chatbots.

Another real world application of a chatbot within the field of cybersecurity constitutes Elastic Endpoint Security's Artemis [20]. Artemis is an intelligent assistant developed to significantly ease the work of cybersecurity experts. The chatbot takes care of the resource consuming analysis of data, freeing up the scarcely available security talents for more complex work. Moreover, it enables experts to write their questions in natural language and eliminates the need for elaborate query languages in order to get the desired answer or data. Artemis has been trained with the experience of various senior security experts who have been active in several sectors, such as the financial, the military or government sector. With this knowledge base, Artemis can efficiently predict the immediate moves and operations based on the actual incident. Consequently, no matter how much experience the cybersecurity experts possess, Artemis contributes to improving the workflow [20]. As we have just seen, Artemis employs NLP and other AI technologies, so that we can add Artemis as a direct consequence to the family of machine learning chatbots.

As we just have seen, there are already some approaches that employ conversational chatbots for the area of cybersecurity, though there is no work that examines chatbots to improve cybersecurity management. In this sense, [23] introduces SecBot, a cybersecurity-driven chatbot, which will be more closely presented in the next chapter.

Chapter 4

SecBot

SecBot is a conversational chatbot that was previously defined by the Communication Systems Group of the University of Zurich. In their scientific paper [23], Franco et al. present a cybersecurity-driven conversational agent aiming towards the simplification of the cybersecurity planning and management of small and medium-sized businesses.
As the introduced conversational agent will be presented in more detail in the following sections, the subsequent content of this chapter is therefore based on [23].

SecBot is especially designed for users with little to no security knowledge. It is able to receive user requests for cybersecurity support and then provides detailed responses which the user then takes into account in the cybersecurity decision-making process.
More specifically, SecBot's capabilities include the identification of cyberattacks on the basis of associated symptoms that the user submits, the specification of solutions and configurations tailored to the user's business requirements as well as giving insightful information to help deciding on cybersecurity investments and risks. To achieve this, SecBot makes use of various concepts from the chatbot development:

- **Intents:** Intents correspond to users' intentions while interacting with a chatbot. Every intention from every user utterance is mapped to an intent.
- **Entities:** Entities are structured information, which can be extracted from user intents. They are bound to knowledge databases, in which acceptable values are given for each entity. This ensures more precise answers.
- **Stories:** Stories represent conversations between humans and chatbots. They require the specification of the necessary steps and based on the input or response of the user, many different conversation paths emerge. This concept is crucial for the model to generalize and thus cover conversation flows that have not been thought about, as well as predicting the subsequent actions correctly.
- **Actions:** Every response from a chatbot to a user input can be considered as an action. More specifically, actions can consist of giving feedback as a response, listening to new utterances, or simply executing user-defined actions, which include custom code for reacting to specific flows.

In order to proof the feasibility of SecBot, the CSG created a Proof-of-Concept (PoC), where for each capability a basic implementation was developed. For this purpose, the Rasa version 1.4.6 was used.

Figure 4.1 illustrates an example of how an intent, which includes multiple entities, is implemented in SecBot. More particularly, it shows the *problem_desc* intent, which should enable SecBot to recognize when a user is reporting a certain problem he is encountering.

```
## intent:problem_desc
- My [files](target) are [encrypted](problem)
- My [server](target) is [overloaded](problem)
- I [cannot access](problem) my [database](target)
- There is a message asking [bitcoins](cryptocurrency)
- I receive a message to pay [ethereum](cryptocurrency)
- My [server](target) is receiving [a lot of requests](problem)
- It is from [different IPs](problem)
- It is [many SYN packets](problem)
- I am experiencing [unexpected popups](problem)
```

Figure 4.1: SecBot's desc_symptom intent [24]

Below the title of the intent, multiple training examples are provided. For each of them, all entity values are indicated in square brackets and their corresponding entities directly behind them in round brackets.

As mentioned before, SecBot makes also use of the concept of Stories, so that the ML algorithms can learn more efficiently, and thus acquire a deeper knowledge regarding the extraction and processing of user input. For that, two approaches were classified: The first one being a *reactive* approach which focuses on scenarios where defending against impending attacks is priority and the second one being a *proactive* approach where the user intends to have a more cost-effective and resilient cybersecurity strategy for his business.

```
## My files are encrypted
* problem_desc{"problem": "encrypted"}
    - slot{"problem": "encrypted"}
    - utter_add_symptom
    - action_symptoms
* deny
    - action_idattack
```

Figure 4.2: SecBot's Files encrypted story [24]

Figure 4.2 shows a story based on the reactive approach. Intents are annotated with an asterisk character and extracted entities are specified within curly brackets. Below each intent, you then specify the sequence of actions which should take place. Since chatbots do not run the action server during training phases, slots must be set manually in stories. In this example, this is done as the first action after SecBot recognized the *problem_desc* intent. The actions *action_symptoms* and *action_idattack* are custom actions which run custom code to process the provided symptoms and possibly identify cyberattacks. Custom actions are written in an external file using the Python programming language. Besides the very basic attack identification story, SecBot also provides basic stories for protection configuration and economic analysis, which in turn also use custom actions.

By using stories to generalize to unseen conversation flows indicates that SecBot follows a machine learning-based approach. On the other hand, stories are basically like hard-coded rules. Based on inbound user intents, SecBot is trained to respond with an appropriate action. This behaviour, in turn, is based on the rule-based approach. Thus, recalling Table 2.4, we can clearly classify SecBot as a hybrid chatbot model by leveraging features of both concepts.

As introduced in the previous paper [23], SecBot provided us with a PoC, thereby giving us the opportunity for many improvements. The aim of this thesis now is to enrich the prototype with new features and knowledge (e.g. Intents, Entities, Stories, Custom Actions, etc.), and thus provide a working prototype that generates accurate responses to user demands.

Chapter 5

Approach and Refinement

As mentioned in the previous chapter, the actual prototype of SecBot does not provide a fully fledged out implementation. The available code clearly only serves to proof the feasibility of the proposed conversational agent. So in that sense, the state of the prototype gives us the opportunity for numerous improvements.
For that purpose, there are generally three broad dimensions which can be considered for further expansion: cyberattack identification based on provided symptoms, specification of solution and configurations, as well as cybersecurity investments support. For each of these dimensions, the enhancement includes the development of new features, i.e. new intents, entities, stories, etc. in addition to new scenarios.

However, due to time constraints, I will not be able to address all of the three dimensions and must therefore narrow down the scope of the thesis. For this reason, I have decided to set my focus on two features that will enhance SecBot's usability in delivering cybersecurity support, especially for non-security experts.
The first feature to be addressed, the symptom description feature, aims to correctly extract and process symptoms from user input in which users describe the problems they are experiencing. As part of SecBot's attack identification capability, the elaboration of this feature is vital to the chatbot's ability to effectively capture the required information.
The second feature to be covered is an attack support feature, which aims to provide users with general information about certain attacks such as a general description, differences between two similar attacks, or common protection measures.
In addition to the development of these two features, the optimization and configuration of the training data set constitutes a third subgoal, which enables the model to be trained more efficiently and thus achieve higher accuracy in understanding the NLU data and predicting the next actions.

The remainder of this chapter provides a non-technical overview of the approach, new features, and how these new features are then used to refine SecBot or provide new capabilities.

5.1 Initial Setup and Configurations

First of all, I had to deploy SecBot from the respective GitLab repository [24] to my computer. The prototype was developed using the Rasa version 1.4.6. Since Rasa is a newer framework, there have been and will be major changes in new versions that will impact the development of conversational chatbots and SecBot in particular.

So was it with the introduction of the Rasa Open Source release candidate 2.0.0rc2 just shortly after the start of the thesis. This release candidate was basically a preliminary version of Rasa 2.0 that introduced major changes, such that all of the code implemented with Rasa Open Source 1 became legacy code. With this in mind, the migration to Rasa 2.0.0rc2 was inevitable in order to best improve SecBot by taking advantage of the various new or improved features and concepts. It involved as a direct consequence the adjustment of SecBot's configurations file, where deprecated policies needed to be properly replaced, as well as SecBot's training data files, where the data type was changed from markdown to yaml.

Despite the many improved and also new features, there were also many drawbacks. The Rasa team has failed to provide detailed documentation and tutorials for the novelties and enhancements right from the beginning, which may result in newbies, including myself, missing on important information during development. Moreover, these many changes led to various bugs in Rasa's source code that could be only continuously fixed with new releases. That said, the continuing development of SecBot included the task of always keeping an eye out for new versions and migrating accordingly in order to best improve the conversational agent.
Since the command line interface (CLI) was least affected by the bugs, training the dialogue management model and chatting with the bot has been done through Rasa's CLI.

5.2 New Features

5.2.1 Attack Support Feature

As mentioned before, this feature provides general information about certain cybersecurity attacks that have been covered in Chapter 2.1. It includes new and different conversation flows such as requesting general information, comparing similar attacks with each other (e.g. computer worm vs. computer virus) or providing information about common protection measures. With this in mind, besides providing the functionality in the code, I also needed to add more knowledge to the chatbot. For this purpose, in addition to the information provided in Chapter 2, I have mapped the information of the subsequent tables according to the concepts of Intents, Entities, Stories, Actions, and so on. For each attack discussed in Chapter 2.1, the tables list the apparent symptoms when an infection or an actual attack occurs and the corresponding effects of such infections or attacks. Even though this scenario is somewhat intertwined with the dimension of supporting cybersecurity investments and risks, it is nonetheless a new feature added to SecBot.

Table 5.1 presents general symptoms and the impact on businesses specifically regarding DDoS threats. As you can see, there are quite a few symptoms that indicate a possible DDoS attack.

Table 5.1: Symptoms and Impacts of DDoS Attacks

<table>
<tr><th colspan="2">DDoS Attack</th></tr>
<tr><th>Symptoms</th><th>Impact</th></tr>
<tr><td>

Website or service becomes unexpectedly slow.
Website or service becomes all of a sudden unavailable.
Unable to access a website in the long term.
Suspect amount of network traffic either from one IP address or a range of different IP addresses.
Inbound traffic flood from users with similar behavior patterns, including device type, location or simply the version of web browser.
An inexplicable increase in requests to a website or endpoint.
Unusual traffic flows (e.g. spikes at odd times of the day or spikes all 20 minutes, etc.).
Extremely slow access to files.
Increasing amount of spam emails, trying to overload the network.
Slow servers responding in minutes instead of seconds (could also be the old equipment or a natural spike from a news event).
Slow computing performance in general (e.g. taking longer as usual for routine tasks).

</td><td>

Website breakdown: As long as your website is not available it will harm your reputation since other companies will not be able to access it as well. It could also influence the search ranking in a negative way.
The attack on one server might have a negative impact on several other websites as well.
Rising vulnerability of websites: Such attacks may only serve to make the website or server more susceptible to further attacks, such as hacking or data theft, as all resources are dedicated to getting everything back up and running.
Lost time and money: In a worst-case scenario where backups are not available, companies might end up hiring experts to completely recreate the entire website from the ground up, highlighting security issues for future cyberattacks. Furthermore, in industries such as e-commerce, website downtime would cause a huge loss of revenue.

</td></tr>
</table>

Depending on the severity of the attack, some of the possible effects may be more pronounced than others. In general, however, it is said that the reputational damage suffered by companies is the most serious impact of such attacks [53].

Table 5.2 shows the same characteristics, but tailored to malware attacks. It is worth noting that while DDoS attacks are more of a threat to companies which have a strong presence on the Internet, malware attacks, on the other hand, constitutes a cybersecurity threat to any kind of business.

Table 5.2: Symptoms and Impacts of Malware Attacks

Malware Attack	
Symptoms	**Impact**
• Continuous crashes (e.g. systems, programs, Blue Screen of Death). • Slow computer performance (e.g. sluggish start, slow when loading applications). • Unusual experience when surfing on the Internet (e.g. new toolbars popping up, inexplicable redirects). • Unusual and unfamiliar new desktop icons (e.g. Potentially Unwanted Programs (PUPs)). • Difficulties with the hard drive (e.g. loud noises, excessive activity on computer). • Computer storage is inaccessible or corrupted. • Unusual error messages. • Unwanted and annoying popup messages. • Disabled security program. • No access to critical programs. • Encrypted files or programs.	• Data breaches: Unauthorized access to confidential data. By infiltrating the target's server, sensitive data can be stolen, which can lead to identity theft, amongst other things. • Downtime: Attacks can break down the company's network infrastructure which will have severe effects on its operations. • General disruption or even blocking of services by taking control of system-critical programs. • Long-term damage to the company's data, infrastructure and reputation, leading directly to the loss of customers. • Money loss respectively money extortion

There are some ambiguous symptoms, such as "slow computer performance" or "slow

access to file”, which are listed in Table 5.1 and Table 5.2. Such symptoms could also be caused by several different attacks or simply have a different origin, such as slow Internet connection or old equipment. Nevertheless, these are symptoms that need to be taken into consideration despite the possibility that they do not have an evil origin. This underlines all the more the complexity of such attacks, whose detection is becoming more and more difficult with every passing day.
The optimization of the attack tree algorithm, which is not within the scope of this thesis, would then have to take such symptoms into consideration when investigating a possible attack.

Table 5.3: Symptoms and Impacts of Phishing Attacks

Phishing Attack	
Symptoms	**Impact**
• Inconsistencies in domain names: e.g. received email from a company using a public domain like ”@outlook.com”, or misspelt domain names. • Grammatical (and spelling) errors in emails. • Email contains threats or generates a sense of urgency. • Email including suspicious links or attachments. • Emails asking for login, payment or other personal information. • Unfamiliar tone or greeting (e.g. when family or employees become too formal). • Recipient has not started the communication. • Uncommon requests: E.g. email from IT department sending a link for a security patch for the recipient’s computer.	• Financial damage: In the event of such an attack, on the one hand there are direct costs due to data protection breaches, which amount to approximately 3.86 million dollars, but on the other hand there are also fines to be paid, which are imposed by regulatory institutions. • Reputational damage: Company appears unreliable and unsafe to the public which may lead to the loss of customers. • Loss of enterprise value: The damage to the business’ reputation can cause investors to lose confidence and withdraw. • Intellectual property theft: The compromise of confidential data, such as trade secrets, research results, or formulas as a direct consequence of a phishing attack can result in wasted research investments and a substantial financial loss. • Business disruption occurs when systems must be taken offline until the threat is eliminated.

Table 5.3 introduces the respective information for phishing attacks. Verizon reports that in 2020, phishing attacks were involved in over 60% of all occured data breaches [65]. Since most of these attacks are conducted through email scams, most of the listed symptoms refer to symptomatic phishing signs in emails.

It is now clear what information has to be considered when extending SecBot's knowledge base in order to be able to implement this newly proposed functionality. Therefore, the following sections can from now on focus on a brief introduction of the new corresponding intents, actions, rules, etc. for both the attack information and attack description feature.

5.2.2 Forms

During the analysis of SecBot's initial implementation, I recognized that the entire core logic was based on custom actions, training stories and their generalization. For instance, the symptom capturing process was defined by only two stories, which were then used to train SecBot to recognize, extract, and process the symptoms provided by the user. For simple conversation cases, this is a legitimate approach. However, with more details, the degree of complexity grows immensely. To cover the various flows of multi-turn dialogues, one would need a sheer amount of training stories to train and generalize from, such that the chatbot could be able to almost always extract all the necessary information and react appropriately to unforeseen paths. Thus, the training data would become bigger and bigger.

That is why I introduced forms as a completely new feature to SecBot. Forms represent a conversation pattern that is used to collect important pieces of information in a dialogue with the user. Since forms do store the requested information in slots, form actions are therefore also referred to as slot-filling processes. By leveraging forms, the business logic can be easily implemented in conversational agents.

In this sense, in order to efficiently gather all the necessary information for the attack identification process, I equipped SecBot with two new forms: a *symptom_form* and a *more_info_form*. The first is used when the user initially wants to submit specific symptoms or provide a general description of the problem, whereas the latter is used when the identification of the attack failed with the actual symptoms provided via the *symptom_form* and you want to add more symptoms and then try again.
By using forms for the symptom description scenario, the complexity of the information gathering process, for which I was able to implement a strict logic, is significantly reduced. With this in mind, SecBot is now able to enforce important information from the user and store it appropriately for later use. Furthermore, to cover all happy paths, i.e. conversation flows where the user submits all the necessary information, significantly less training stories are needed, thus shrinking the amount of training data required and hence reducing the training time.

5.2.3 Training Data

When using the Rasa framework, the training data is split into NLU (e.g. intents, entities, etc.) and conversational training data (e.g. rules, stories). To improve SecBot in general, I needed to create additional training data for both subcategories.

Intents

During development I was also able to create new intents. With respect to forms, I created a new intent called *yet_more_info* which is used to trigger the *more_info_form*. It contains several training examples of how a user could ask to add more symptoms to the ones already submitted or simply describe the problem in more detail. For the *symptom_form*, on the other hand, there already existed suitable intents to invoke the form. However, I was able to leverage a new functionality which combines multiple user intentions into a single one. With this in mind, I defined two new so-called multi-intents *greet+attack_notification* and *greet+problem_desc*, which now are also able to trigger the *symptom_form*. These two multi-intents combine the *greet* intent with the *attack_notification* or *problem_desc* intent and are intended to make the conversation more natural. They are used when the user greets and starts complaining within a sentence.

Another intent I have created is the *explain* Intent. This intent is recognized when a user becomes suspicious of SecBot's request for some specific information during an active form. For instance, during a form action, if the user wants to know why some information is needed, then the chatbot will respond accordingly before proceeding with requesting the required data. For the case when the user does not want to provide this information or is simply not able to provide this kind of information, the user should then be able to abort the form action. For this purpose, I have defined a *stop* intent that takes care of such situations in both presented forms.

With respect to the attack support feature, there have been seven new intents defined. The first one, the *request_attack_information* intent, is classified when a user wants to know what a particular attack is respectively how it works. Such questions may arise when SecBot has been successful in identifying an occurring attack and the user knows nothing about that attack. The other use case is that a user might also be talking to SecBot for educational reasons and is interested in knowing the characteristics of a particular attack. Similarly, if the user still wants to know even more details about the specified attack and is curious whether there exist certain types or subtypes, these questions then are defined under the *request_further_attack_classification* intent.
Another scenario is that the user may also want to compare certain attacks. This could be the case if the user has very basic security knowledge and expected, for example, DNS amplification to be identified as the current attack, but SecBot identifies it as a SYN flood. In this case the user is interested in learning what the difference is between these two attacks. Or, for instance, if the user previously asked about other subtypes of malware and the explanations for, say, worms and viruses sound too similar. Such cases require further clarifications in the form of differences.
In addition to the aforementioned intentions, the attack support feature also offers a whole

lot more information. For example, there may be the case that the user asks about the potential symptoms of an attack out of curiosity, or this information is requested to see what other symptoms the user may have possibly missed out on his/her device. To cover these kinds of concerns, I have created the *request_attack_symptoms* intent. Another scenario that can occur is that the user wants to know what kind of challenges the attack poses. As such, this concern has been addressed with the *request_attack_challenges* intent. Consequently, once a general overview over the attack is obtained, further non-technical questions could be asked. One such question revolves around the impact that a particular attack may cause to businesses. These kind of questions are especially important for users who own or are responsible for the IT security of a business and are addressed by the *request_attack_impacts* intent.
Finally, the last intent that comes with the attack support feature involves possible protection measures and is defined as *request_countermeasures* intent. More specifically, if this intent is classified, the user then asks for general protection measures that are used to combat the particular attack.

Moreover, besides dealing with symptoms and question such as those covered in the attack support feature, SecBot should also be able to conduct small talks. This behaviour could be achieved by creating two new retrieval intents. These are special types of intents that consist of multiple smaller subintents. In this sense, I created a *chitchat* retrieval intent that contains several subintents, such as *ask_isbot*, where the user asks SecBot if it is indeed a chatbot, or *ask_howbuilt*, where the user wants to know how SecBot was developed. Further, a second retrieval intent, an *out_of_scope* intent, was added to the NLU data to handle any other out-of-context messages that have not been covered within the *chitchat* intent. However, small talks are intentionally very limited, as the vision is not to make SecBot a chatter bot, but a conversational agent that offers support for cybersecurity issues.
Lastly, I have also included an intent called *nlu_fallback*. This intent is a predefined intent that needs to be explicitly specified, but no further conversation examples are needed for it. It works together as part of a component and a policy, which will be discussed in more detail later.

Rules

As mentioned in previous sections, Rasa 2.0 came with various new or improved features. Rules, as a result of combining multiple rule-based policies within a new general *RulePolicy*, constitute one such new feature which is now part of the conversation training data. They are basically short pieces of conversation that follow always the same structure. They cannot be generalized like stories, but they come in handy when it comes to implementing predictable conversational dialogues with fixed responses.

For SecBot, the use of rules brings more robustness into the dialogue managment model, since this rule-based approach is a complement to the machine-learning one. With this in mind, I have specified rules for the following occasions:

- **Forms:** The process of activating as well as deactivating and submitting a form always follows the same structure. For this reason I have implemented respective rules. Furthermore, I have also specified rules to handle unexpected input, such as out-of-context or chitchat intentions during form actions, thereby covering unhappy paths. Note that the intents addressed by the action support feature are also treated as unexpected input when they occur during form actions.

- **Fallback handling:** In the case of user messages with low NLU confidence, a fallback rule responds.

- **One-turn interactions:** Predefined responses to user utterances where either the context is not relevant, such as chitchat or out-of-scope intents, or where certain intents require particular custom actions to be executed, as it is the case with intents of the attack support feature.

Now, the fact that we are able to explicitly specify rules reinforces the statement in Chapter 4, where we defined SecBot as a hybrid model that leverages the best features of both rule-based and machine learning-based approaches.

Stories

To train SecBot's dialogue management model more adequately, I also had to create more stories for the conversational training data. Altough I have introduced rules, they cannot be generalized. In this sense, explicit conversation examples covering various conversation flows had to be provided so that SecBot could learn the association between intents and bot responses and thus also generalize from them. With this in mind, I have first covered all conversation flow where the user submits all requested information, i.e. happy paths, for both the *symptom_form* and the *more_info_form*. Up next, I wrote stories for unhappy paths during active forms. These included stories where requested information was missing or unexpected messages including chitchat, out-of-scope, or questions addressed by the attack support feature were given as input.
Moreover, I have also created additional stories for the unhappy path during the form actions that cover contextual interjections, i.e. unexpected input whose response depends on the current context. These kind of conversation samples basically train the assistant on how to react to the previously introduced explain intent.
Finally, once a certain amount of features was implemented, I started to create stories for the unhappy paths using Rasa's interactive mode. This gave me the opportunity to create more sophisticated conversation examples and also ensured that all necessary slots were set, which was not as trivial as it sounds with the increasing complexity of the conversation samples.

5.2.4 Actions

The first type of action I want to discuss are custom actions. With regard to the symptom description feature I implemented five new custom actions. First, in order to obtain the correct values, the user input must somehow be checked for validity during the form

actions. To do this, I have created two custom actions, *ValidateSymptomForm* and *ValidateMoreInfoForm*, which are responsible for validating all the extracted slot values from the corresponding user messages during their respective form actions. Once a form has been completed it must then be submitted to be able to process the information. For this reason I have defined the *ActionSubmitSymptomForm* and *ActionSubmitMoreInfoForm* actions that take the responsibility for submitting the respective forms after all required information has been gathered. For the event that a form is interrupted, the already extracted slots need to be reset. Otherwise, when entering the form in a later stage, there will be an information conflict as the already extracted slots will not be refilled. For this reason, I have implemented the *ActionResetSlotsAfterFormInterruption* action, which takes care of resetting the corresponding slots.
With respect to the attack support feature, for each of the intents discussed, a corresponding custom action is implemented.

The second type of actions denote bot responses. For all the newly created training data, several new responses with different variations have been implemented, allowing SecBot to react appropriately in any situation. A complete list of responses is available as Appendix C. The last type of actions are form actions. Since forms are implemented as a completely new feature in SecBot, it has already been discussed previously in Subchapter 5.2.2.

5.2.5 Slots

In order to make SecBot more context-aware, various new slots needed to be defined. Table 5.4 outlines the new created slots with a short description of what values they store.

Table 5.4: New Slots

Slot	Description
identified_attack	Stores the last attack that was identified by SecBot
symptoms_1	Stores the symptoms submitted during the *symptom_form*
symptoms_2	Stores additional symptoms submitted during the *symptom_form*
symptoms_target	Stores the target specified during the *symptom_form*
more_info	Stores the additional submitted symptoms during the *more_info_form* after the attack detection failed the first time.
new_target	Stores the new target during the *more_info_form*.
requested_slot	Stores all slots of *symptom_form* and *more_info_form*.
attack_information	Stores the last attack(s) for which a description was provided.

further_attack_classification	Stores the last attack(s) for which further classification was provided.
compared_attacks	Stores the last attacks that were compared with each other.
attack_challenges	Stores the last attack(s) for which challenges were provided.
attack_impacts	Stores the last attack(s) for which impacts were provided.
attack_symptoms	Stores the last attack(s) for which potential symptoms were provided.
attack_countermeasures	Stores the last attack(s) for which common protection measures were provided.

5.3 Refinements

5.3.1 Training Improvements

The first refinement step regarding the configurations of SecBot already started with the migration to Rasa's release candidate. For that, I removed three policies (*MappingPolicy*, *FormPolicy* and *KerasPolicy*) which have all become deprecated. The *MappingPolicy* and *FormPolicy* were redundant anyway, since their functionality (e.g. forms) were not implemented. Then, as a solution to the deprecated *KerasPolicy*, I added the *TEDPolicy* within the policies section instead. It is a machine learning policy that is used to generalize from the training stories such that SecBot is also able to cover unforeseen conversation paths like off topic messages.
In order to make use of forms and to specify rules, the *RulePolicy* was added to the policy section. It combines the functionality of several rule-based policies into one policy, thus making the configuration of the policies section more approachable.

Furthermore, I have also made some changes to the component pipeline which is responsible for turning user messages in form of unstructured data into structured data so that the chatbot is able to understand the user's intentions. To facilitate better intent classification and entity extraction, I have now added two new featurizers, namely the *LexicalSyntacticFeaturizer* and the *CountVectorsFeaturizer* respectively.
Moreover, from the initial pipeline configuration I replaced two components, the *CRFEntityExtractor* and the *SKlearnIntentClassifier*, with the so called *DIETClassfier*, which combines Intent classification and Entity extraction within one component. The reason for sticking to the *DIETClassifier* is that, besides leveraging multi-intent classification, it offers great flexibility due to its highly customizable nature. You can also use it only for intent classification or just for entity extraction. Furthermore, there is the possibility to include different pre-trained word embeddings or simply stick to your trainings data.

To fine-tune the training, the *DIETClassifier* allows even more tweaking of settings with the various hyperparameters that can be set, reinforcing the component's customizability. Based on state-of-the-art architecture, models can be trained up to six times faster with *DIETClassifier*, as described in [39].
Moreover, I have added two more *ResponseSelector* components that are responsible to correctly reply with predefined responses if a retrieval intent is recognized. Also for the previous mentioned fallback handling, the corresponding component, the *FallbackClassifier*, had to be included into the component pipeline.

With all these refinements made, I was able to further fine-tune SecBot's configuration by setting and adjusting some of the parameters that the policies and components provide. As this information becomes more technical, these settings will be covered in more detail in Chapter 6.

5.3.2 Training Data

Here again, the refinement of the training data already started when I was migrating SecBot to Rasa's newest release. What changed was the data format from markdown to yaml type for both the NLU and the conversation training data. Therefore, the training data had to be adjusted accordingly.

Furthermore, the newly created intents discussed in Subchapter 5.2.3 provide SecBot with a whole set of additional training examples which are used to accurately train the dialogue management model. In addition, the investigation on cybersecurity threats in Subchapter 2.1, along with the additional information from the attack support feature (cf. Subchapter 5.2.1), allowed me to adequately refine the actual knowledge base. To this end, I added more examples for several intents, with the *problem_desc* intent benefiting the most, as this intent is crucial to the symptom description scenario.
What's more, the refinement of the knowledge base also included the refinement of so-called lookup tables, which serve as an aid to accurately identifying the user intentions by creating patterns using case-insensitive regular expressions.

With regard to the existing stories, the refinement was a continuous process throughout SecBot's development. After the introduction of the form feature, various stories had to be rewritten and therefore adjusted to the new functionality. Once writing stories got more complex, mainly due to several newly introduced slots, I often switched into Rasa's interactive mode to rewrite an existing conversation example.

5.3.3 Custom Actions

In terms of actions, there was not much to refine. Since the attack information is a completely new functionality added to SecBot, no other existing custom actions were affected, except for the *ActionIdAttack*, which is responsible for identifying the attack based on the given symptoms. There I only made a small change so that the algorithm sets the new *identified_attack* based on its result.

In the case of the symptom description scenario, the new form feature including the corresponding custom actions took over the responsibility for collecting symptoms from the *ActionSymptoms* action. I therefore removed this action completely. Another action called *ActionPardon* was also removed, since it didn't contribute to any functionality discussed in this report. It was originally used for understanding the functionality of certain other classes such as *Tracker* or *Dispatcher* by printing out the outcome of important methods.

Chapter 6

Implementation

Complementary to Chapter 5, this chapter provides a more technical overview of the implemented and refined features. First, Chapter 6.1 takes a closer look at SecBot's configuration before discussing the implementation of the symptom description feature in Chapter 6.2 and the attack support feature in Chapter 6.3 in more detail. Then, Chapter 6.4 shows the process of how SecBot has been integrated into the popular messaging application Telegram.

As for the technology stack used, SecBot is build entirely with Rasa's open source machine learning framework. Since it is a newer framework, new versions are constantly being released to fix bugs that have occurred or to generally enhance the framework itself. For this work, Rasa version 2.2.2 and Rasa SDK 2.2.0 are currently used.
Rasa's action server, which is responsible to execute custom actions, is based on the Python programming language. During development, Python version 3.7.5 has been used throughout.

6.1 Configuration

The configuration of SecBot is defined in the configuration file *config.yml* and consists of two main parts: the component pipeline and the policies section.

The components are responsible to convert the unstructured user input into a structured form consisting of intents and entities, thus enable the assistant to correctly capture the user's intention. This process happens sequentially, i.e. starting from the top component and ending with the last one, which at the same time means that the order matters. Thus, every component, except the first one, gets as an input the output of the previous component.

Listing 6.1 illustrates how the NLU pipeline is structured. On top, *SpacyNLP*, an open-source NLP library, is initialized in order to use *spaCy* components.

```
1 # https://rasa.com/docs/rasa/nlu/components/
2 language: "en"
3
4 # Configuration for Rasa NLU.
5 pipeline:
6 - name: "SpacyNLP"
7 - name: "SpacyTokenizer"
8   intent_tokenization_flag: True
9   intent_split_symbol: "+"
10 - name: "SpacyFeaturizer"
11 - name: "RegexFeaturizer"
12 - name: LexicalSyntacticFeaturizer
13 - name: CountVectorsFeaturizer
14 - name: CountVectorsFeaturizer
15   analyzer: "char_wb"
16   min_ngram: 1
17   max_ngram: 4
18 - name: "DIETClassifier"
19   epochs: 400
20   use_masked_language_model: True
21 - name: "EntitySynonymMapper"
22 - name: "ResponseSelector"
23   epochs: 70
24   retrieval_intent: chitchat
25 - name: "ResponseSelector"
26   epochs: 70
27   retrieval_intent: out_of_scope
28 - name: "FallbackClassifier"
29   threshold: 0.3
```

Listing 6.1: Configuration - Component Pipeline

SpacyTokenizer takes the initial user message and splits it into tokens. *SpacyFeaturizer*, on the other hand, creates features, i.e. vector representation of the user message, which then are used for classifying intent and responses, as well as extracting entities. The *RegexFeaturizer* also provides features for the same purposes, but in the case of SecBot, this featurizer is mainly used to leverage lookup tables that provide help for correctly extracting entities that have many possible values.
To enable multi-intent classification, I have had to set the corresponding hyperparameters within the *SpacyTokenizer*. More specifically, I set the *intent_tokenization_flag* to *True* and specified the delimiter string for the *intent_split_symbol* parameter. An example of a multi-intent illustrates Listing 6.6 in Chapter 6.2.
As the name indicates, the *LexicalSyntacticFeaturizer* on line 12 provides lexical as well as syntactic features to support the process of entity extraction. This component is followed by two instances of *CountVectorsFeaturizer*, which are typically used to provide features for classifying intents and selecting responses by creating bag-of-words representations of user message, intent, and response. The first instance on line 13 is therefore responsible for counting whole words, whereas the subsequent instance is configured to look at sub-word sequences of characters, also known as character n-grams. For this purpose, the parameter *analyzer* must be set accordingly and the lower and upper limits of the n-gram must be configured as with the parameters in lines 16 and 17, respectively.

Not all intent classifier and entity extractors are designed to handle multi-intent classification, but the *DIETClassifier* I introduced in Chapter 5.3 offers this capability. To fine-tune the training, this component offers various hyperparameters that can be specified. In Listing 6.1 you can see that the *epochs* parameter on line 19 is set to 400 and the *use_masked_language_model* to *True*. *Epochs* denote the amount of times the training goes through all of your data. 300 is the default value and depending on the size of your training data this parameter needs to be adjusted. Otherwise, your model might either generalize too much, or in the other case, it learns the data so well that it starts to memorize rather than generalize. *Use_masked_language_model*, on the other hand, needs to be set to *True* if your assistant needs to acquire additional domain knowledge through your data. This is the case, for instance, if you expect long or complex input messages, the domain language contains slight variations, or the number of intents grows. With respect to SecBot, this parameter must be set to *True* especially for utterances which do not contain any entities. An example of this is the following: Depending on whether the user asks *"What impacts does this attack have?"* or *"What subtypes does this attack have?"*, the assistant classifies the message either as a *request_attack_impacts* intent or a *request_further_attack_classification* intent.

The *EntitySynonymMapper* component maps equivalent entity values, such as distributed denial-of-service and DDoS, to the same value. For the two *ResponseSelector* components, the name of the retrieval intent must be specified and optionally the number of epochs. If the results of the intent classification are ambiguous, then the *FallbackClassifier* classifies the message with the *nlu_fallback* intent. This happens when the intent classifier could not categorize the intent with a probability greater than or equal to the threshold which is currently set to 0.3.

Listing 6.2, on the other hand, illustrates the policies section which is responsible for predicting the next actions. In the case of multiple policies predicting the next action with the same probability, the default policy priority ranking is then taken into account.

The *MemoizationPolicy* recalls all implemented stories and compares them to the actual conversation. It either predicts the next action with a probability of 1 (meaning there exist a story matching the current conversation) or in the other case, it predicts *None* with a confidence of 0.0.
The *TEDPolicy*, however, is used for generalization purposes. In this sense, it takes into account multiple factors, such as user input, previous actions or bot utterances, as well as slots and active forms, before making predictions about the next action. For that purpose, the hyperparameter *max_history* can be set, which determines how much dialogue history will be considered. The default is considering the complete history. The *epochs* parameter can also be set here. In this case the default value is set to 1, but must be adjusted depending on the model to learn appropriately. With regard to SecBot, this parameter is set to 80.

```
# Configuration for Rasa Core.
# https://rasa.com/docs/rasa/core/policies/
policies:
  - name: MemoizationPolicy
  - name: TEDPolicy
    # max_history: 10
    epochs: 80
```

```
 - name: RulePolicy
   core_fallback_threshold: 0.3
   core_fallback_action_name: "action_default_fallback"
   enable_fallback_prediction: True
```

Listing 6.2: Configuration - Policies

The *RulesPolicy* combines all rule-based policies (for rules, forms, fallback, and so on) into one component. With respect to the fallback behaviour, three more parameters need to be set. The *enable_fallback_prediction* flag must be set to *True*. This then allows the *core_fallback_threshold* parameter to be set to a certain confidence score. Each time the machine learning policies are unable to predict an action with a probability greater than the *core_fallback_threshold*, the action specified by the *core_fallback_action_name* parameter, in this case the *action_default_fallback*, is then executed.

6.2 Symptom Description Feature

Chapter 5 provided a high-level overview of the new and improved features that should, among others, bring some robustness to the process of describing the symptoms and problems a user is experiencing. This chapter, however, provides a more in-depth and technical view of the implementation of these features.

For starters, we will take a closer look at the brand new forms. Forms are created within the *domain.yml* file by adding the form under the *forms* section. Listing 6.3 shows the implementation of the *symptom_form*.

```
forms:
  symptom_form:
    symptoms_1:
    - entity: problem
      type: from_entity
    - intent: deny
      type: from_intent
      value: None
    symptoms_2:
    - entity: problem
      type: from_entity
    - intent: deny
      type: from_intent
      value: None
    symptoms_target:
    - entity: target
      type: from_entity
    - intent: deny
      type: from_intent
      value: None

```

Listing 6.3: Symptom Form

```
  more_info_form:
    more_info:
    - entity: problem
      type: from_entity
    - intent: deny
      type: from_intent
      value: None
    new_target:
    - entity: target
      type: from_entity
    - intent: deny
      type: from_intent
      value: None

```

Listing 6.4: MoreInfo Form

As on line 2, the name of the form must first be defined. This is at the same time the name of the action used in stories and rules. Below the form name, the respective slots to be filled, in this case *symptoms_1*, *symptoms_2* and *symptoms_target*, need to be defined. These slots were also previously created in the *domain.yml* file, but under the *slots* section and are of type *list*. It is also necessary to specify for each of the slots how to assign the correct value to it. The requested slots of the *symptom_form* use two variations of slot mappings: *from_entity* and *from_intent*. The first fills the requested slot based on the extracted entities. More specifically, if the user message contains a *problem* entity, then that entity is automatically mapped to the slot. Otherwise, if the intent of the message is *deny*, then the slot value is set to *None*. It is worth noting that if an *target* entity is also extracted while the information for the *symptoms_1* slot is requested, then the *symptoms_target* slot is also filled directly. Consequently, the user won't be asked to explicitly specify the target again. Analogously, Listing 6.4 presents the corresponding implementation for the *more_info_form*.

Form activation happens either through rules or through stories. Since the *symptom_form* should be always activated, whenever the user intends to provide a symptom respectively problem description, activating the form is handled through the rule illustrated in Listing 6.5.

```
1 - rule: start symptom form
2   steps:
3   - or:
4     - intent: problem_desc
5     - intent: attack_notification
6     - intent: greet+attack_notification
7     - intent: greet+problem_desc
8   - action: symptom_form
9   - active_loop: symptom_form
```

Listing 6.5: Rule - Activate Symptom Form

In order to be able to create rules, the *RulesPolicy* must be specified in *config.yml*. Moreover, I also had to create a new *rules.yml* file that is persisted in the same directory as the *nlu.yml* and *stories.yml* training data. All new rules can now be defined below the *rules* keyword located at the beginning of the created file. Listing 6.5 shows how to create such a rule. At the top, you first describe the name of the rule. Then the corresponding steps are defined one after the other.
With respect to the activation of the *symptom_form*, the or-statement describes that whenever SecBot classifies an intent as one of those listed beneath the or-statement, the *symptom_form* action is triggered. The last step in line 9 indicates that the form should be active after the aforementioned form action is executed.

Indeed, the intents *greet+attack_notification* and *greet+problem_desc* are multi-intents that were not available before. An example of the latter is shown below in Listing 6.6. It combines the *greet* with the *problem_desc* intent and allows now SecBot to understand messages that include more than one intent, which also makes the conversation more natural. Generally comparing this example with Figure 4.1 in Chapter 4, you can see that the way how intents are created has changed. Specifying the name of the intent after the *intent* keyword on the first line and defining the examples below the *examples* keyword now significantly improves the readability for the developer.

```
- intent: greet+problem_desc
  examples: |
    - Hello. My [files](target) are [encrypted](problem).
    - Hi my [server](target) is [overloaded](problem).
    - hey SecBot! I [cannot access](problem) my [database](target)
    - Yo, there is a message asking [bitcoins](cryptocurrency)
    - Hello, I receive a message to pay [ethereum](cryptocurrency)
    - Good morning. My [server](target) is receiving [a lot of
     requests](problem).
```

Listing 6.6: Multi-Intent - Greet + Problem Description

Once the form has been activated the user will be prompted to deliver the requested information. This is done in the following way: The assistant looks out for responses which are defined within the *domain.yml* file under the *responses* section. There, for each required slot a response in the format of *utter_ask_<slot_name>* is defined. So, for instance, the response for the required slot *symptoms_1* is defined as *utter_ask_symptoms_1* with SecBot then uttering *"Can you describe what the problem is?"* once the form is activated.

After the user entered their message, the extracted slots have to be validated. Rasa validates by default only if any slot has been filled after a slot has been requested. That is why I have created a custom action to validate the extracted slots. Custom actions in general are defined within the *action.py* file. However, the return value of the *name* method must be added into the domain file under the *actions* section. Otherwise, the assistant won't know that these actions exist and raise errors if they are used in specific rules or stories. Nevertheless, this custom action is only executed if the extracted entity is of type *problem* or the user message is classified as a *deny* intent as specified in Listing 6.3. For all the other user inputs that didn't fill the requested slot the form action will be rejected, thus, an error is raised.

```
class ValidateSymptomForm(FormValidationAction):

    def name(self) -> Text:
        return "validate_symptom_form"

    def validate_symptoms_1(
        self,
        slot_value: List,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ):

        # print("VALIDATING SYMPTOMS_1 SLOT")
        if slot_value == 'None':
            dispatcher.utter_message(template="utter_explain_symptoms_1"
  )
            dispatcher.utter_message(template="utter_stop_form")
            return {"symptoms_1": None}
        else:
```

```
20              # print("SUCCESSFULL VALIDATION")
21              return {"symptoms_1": slot_value}
22
23      def validate_symptoms_2(...):
24          [...]
25
26      def validate_symptoms_target(...):
27          [...]
```

Listing 6.7: Custom action - Validate Symptom Form

In the other case, if the user does not want to or is unable to describe the problem or identified symptoms, SecBot will first explain why it is important to provide those information using the *utter_explain_symptoms_1* response action, as shown in Listing 6.7 on line 16, followed by the *utter_stop_form* action which basically informs the user how to proceed when no symptoms can be provided or the user no longer wants the form to be executed. In this sense, the required slot is set to *None*. This means that it has not been filled correctly. As a consequence, SecBot repeats the question. Otherwise, the extracted information are assigned to the *symptoms_1* slot and the assistant proceeds with querying the other required information.

When all required slots are filled, the form is automatically deactivated and listens to the user for the next user input by default. In the case of SecBot, a submit method is triggered after the form is deactivated. So, since this behavior is to be set as default, this is also achieved by using rules. Listing 6.8 shows how such a rule is designed. Since the form needs to be active in order to deactivate it, the *active_loop* parameter must be set under the *condition* keyword to the form name, in this case to *symptom_form*, to indicate that the current form is activated. To disable the form the *symptom_form* action is executed again, before setting the *active_loop* to *null*. And since we want to submit the form subsequently, the custom action *action_submit_symptom_form* is triggered as the last action in this rule. Moreover, an additional flag called *wait_for_user_input* had to be set to *false*. By default, each rule has this flag set to *true*, which implicitly defaults to *action_listen* as the next action, meaning that the assistant will wait for the next user input. However, this conflicted with several other stories that performed a different action after the submit method.

```
- rule: submit symptom form
  condition:
    - active_loop: symptom_form
  steps:
  - action: symptom_form
  - active_loop: null
  - action: action_submit_symptom_form
  # Every rule implicitly includes a prediction for ‘action_listen‘ as last step.
  # This means that Rasa Open Source will wait for the next user message
  wait_for_user_input: false
```

Listing 6.8: Rule - Submit Symptom Form

The *action_submit_symptom_form* is a custom action which serves for two purposes. First, the *symptom_form* disables itself after it has filled all the required slots. However, the filled slots are not reset to *None*. As a direct consequence, even tough the form is reactivated through the appropriate intents, the user will not be able to enter any messages since the required slots are, in principle, already filled. Therefore, the activation is directly followed by the deactivation of the *symptom_form*. Actually, this would still satisfy our requirement to offer the user the possibility to provide the information in a pleasant and robust way. Though, this would introduce the inconvenience of having to restart the server every time the user encounters a new problem. This issue is therefore addressed within this custom action. More specifically, at the bottom of Listing 6.9 (line 15 and 20), you can see that every time the form is deactivated and thus also submitted, the required slots are automatically reset to *None* so that the user is no longer forced to permanently restart the server. Instead, the user can now restart the form as many times as they want during the conversation.

The second purpose is somewhat intertwined with the attack identification process, which is usually outside the scope of this work. Though, the custom action that is responsible for the attack identification extracts the new symptoms provided from the *problem* slot. At this point, it is important to mention that Rasa has an auto-fill functionality for slots that are named exactly the same as an entity. And this is exactly the case for the *problem* slot. So even if we provide the necessary information to fill the *symptoms_1* slot, for example, the *problem* slot will automatically be assigned the same value. If you then provide the information for the *symptoms_2* slot, the *problem* slot will be overriden with the newly provided entity value. Moreover, it is also not possible to retrieve the required slots of the form as they are now reset after the form is submitted, and since it is also not possible to append values to a slot, even if it is of type *list*, we have to manually update it through this custom action. Otherwise we would be collecting the information during the form action for nothing. For this reason, as shown in Listing 6.9, we collect all the provided symptoms from the required slots into one *symptoms* list, which is initialized on line 7, and set the *problem* slot at end to that list. I've also included a checkpoint to see whether the form worked properly. If the *symptoms* list is empty, then only the required slots are reset.

```
1  class ActionSubmitSymptomForm(Action):
2      def name(self):
3          [...]
4
5      def run(self,...):
6
7          symptoms = []
8          symptoms_1 = tracker.get_slot("symptoms_1")
9          symptoms_2 = tracker.get_slot("symptoms_2")
10         symptoms_target = tracker.get_slot("symptoms_target")
11
12         [...]
13
14         if len(symptoms) != 0:
15             return [SlotSet("problem", symptoms), SlotSet("target",
16                     symptoms_target), SlotSet("symptoms_1", None),
17                     SlotSet("symptoms_2", None),
18                     SlotSet("symptoms_target", None)]
19         else:
20             return [SlotSet("symptoms_1", None), SlotSet("symptoms_2",
```

```
                    None), SlotSet("symptoms_target", None)]
```

Listing 6.9: Custom action - Submit Symptom Form

As mentioned earlier, user may deviate from the happy path for any reason. They may want to ask certain questions, stop the form action or they just want to do some chitchat to see to what extent the assistant can handle unhappy paths (cf. Subchapter 5.2.3). This in turn means that SecBot must be prepared for such unexpected inputs. To make SecBot more resilient in this regard, I made use of both rules and stories. In the case of simple chitchat or completely out of scope messages where the response should always be the same irrespective of the context of the conversation, I leveraged the rules functionality. For that, I first have to assure that the form is active as shown on line 3 in Listing 6.10. Then, when classifying a chitchat intent, SecBot automatically responds to that chitchat message before executing the form action again (line 7) and setting the *active_loop* to the respective form name.

```
1 - rule: return to symptom form after chitchat intent
2   condition:
3     - active_loop: symptom_form
4   steps:
5   - intent: chitchat
6   - action: utter_chitchat
7   - action: symptom_form
8   - active_loop: symptom_form
```

Listing 6.10: Rule - Handle unexpected Chitchat input

In Chapter 5, the *chitchat* and *out_of_scope* intent were introduced as retrieval intents. These special types of intents are divided into subintents and specified in the *nlu.yml* file like all other intents. They can be distinguished from usual intents by their format, which is as follows: *retrieval_intent/sub_intent*. For instance, an actual subintent of *chitchat* constitutes *ask_isbot* which includes examples like *"Are you really a bot?"*. In the *nlu.yml* file, the retrieval intent is then specified as *chitchat/ask_isbot*. Moreover, the corresponding response action is defined as *utter_chitchat/ask_isbot* under the *responses* section which is located in the *domain.yml* file. The advantage of such retrieval intents is that even though each subintent is implemented as an individual intent, they are all handled in the same way during the conversation. This means that, as in line 5 in Listing 6.10, only the retrieval intent, in this case *chitchat*, needs to be specified. The *RulesPolicy* along with the *ResponseSelector* then take over and respond with the appropriate answer depending on what question is asked. The rule of the *out_of_scope* retrieval intent is implemented analogously.

In the case of contextual interjections, i.e. unexpected input whose response depends on the current context, I utilized stories. More specifically, I focused on questions where the user is curious about why a particular piece of information is needed. In Rasa, a *requested_slot* slot of type *text* is implicitly added to the domain and is only considered when a form is active. To be able to answer such contextual questions during a form action, the previously mentioned slot must be explicitly defined in the *domain.yml* file. Further, its type must be set to *categorical* and all values that this slot can take must be specified as well. For our purpose, we include all required slots from both the *symptom_form* and

the *more_info_form*. Moreover, in order to create such stories an *explain* intent has been added to the NLU training data and for each of these slots given as values, an explanation response has been introduced in the corresponding section of the *domain.yml* file.
In Rasa, stories are created inside the *stories.yml* file. As shown on line 1 in Listing 6.11, the name of the story is initialized before a sample conversation under the *steps* keyword is given. These sample conversations are represented through users' messages classified in intents and the assistant's actions, which include both custom actions and defined responses. It must also be pointed out that during training the action server is not accessed. This means that the assistant doesn't know what events are returned during training. Therefore, events such as setting a slot or activating or deactivating a form must be explicitly written inside the stories. For instance, the *slot_was_set* event in line 6 in Listing 6.11 shows that the *requested_slot* slot is automatically set to the *symptoms_1* slot after the form has been activated. If now the user asks the bot why the information is needed, in this case why the user should state symptoms or describe the problem (represented on line 8), SecBot first retrieves the requested slot to see for which slot the explanation is asked before responding with the *utter_explain_symptoms_1* action. Then, with the action that is executed on line 10, the assistant returns back to the form and asks again for the necessary information. Analogously, for each value of the *requested_slot* slot such a story has been created, so that SecBot is now able to always give the appropriate explanation during a form action.

```
1  - story: symptoms_1 interjection
2    steps:
3    - intent: attack_notification
4    - action: symptom_form
5    - active_loop: symptom_form
6    - slot_was_set:
7      - requested_slot: symptoms_1
8    - intent: explain
9    - action: utter_explain_symptoms_1
10   - action: symptom_form
```

Listing 6.11: Story - Contextual Interjection

For the case where the user does not want to provide any information or just want to stop the form, a few more stories have been created. During the form action, if a message is classified as a *stop* intent SecBot then responds with the *utter_ask_continue* action. It basically asks whether the form action should be continued or not. If again a *stop* or a *deny* intent is classified, a default action called *action_deactivate_loop* is executed and the *active_loop* is set to *null*. After that, the custom action *ActionResetSlotsAfterFormInterruption* is directly executed for the reasons discussed in Subchapter 5.2.4, thus resetting all slots after the form has been deactivated and then confirming with the *utter_successfully_stopped_form* response to the user that the form has been deactivated successfully. The *utter_ask_continue* action serves as an assurance to SecBot to verify that the *stop* intent was not incorrectly predicted by the dialogue management model, but was really the user's intention. Because, in case of an incorrect prediction, the user then can only affirm the continuation of the form action and proceed as usual. Note that such an approach could not be implemented by rules and would trigger an error. The confirmation step turns this part of the conversation into a multi-turn dialogue, which is best represented by stories. Implementing it through rules is still only feasible if you omit

the confirmation step and risk mistakenly terminating the form and thus also harming the user experience.
Moreover, all the features including the various stories, rules, custom actions, etc. that were presented in the listings are also implemented anologically for the *more_info_form*.

6.3 Attack Support Feature

As in Chapter 6.2, this chapter provides a more detailed and technical overview of the implementation of the attack support feature and its characteristics.

At first glance, this feature might seem like a basic FAQ feature. In that sense, one would strive for an implementation which relies heavily on retrieval intents. That being the case, subintents for questions such as ”Are there any subtypes of DDoS attacks”, ”What does a DDoS attack do?”, ”What are the impacts of DDoS attack”, etc. must be created for each major attack, along with the corresponding responses. Then, for example, for each subtype of the major attack, the same questions need to be represented again by even more subintents and responses. You can clearly see that such an approach requires a sheer amount of subintents and responses to be created. If there were only two major cyberthreats, say DDoS and Malware, this approach would easily feasible. But that's not the case. There exist thousands of cyberattacks, and even if you wanted to cover only a fraction of them, this approach would blow up the number of training data. Consequently, training the dialogue management model would take substantially longer. Moreover, this approach would only work for one-turn interactions. Follow-up questions, such as ”Does this attack have any subtypes”, where the message include no entities and thus heavily rely on the current context would not work at all. For these reasons, I did not consider the basic FAQ approach, but instead pursued a more context-aware approach whose solution can be easily extended to include additional cyberthreats.

To start with, Listing 6.12 shows how the new *request_attack_information* intent, which was described in Chapter 5, is implemented. As you can see from line 8 downwards, the examples each contain the requested attack, therefore making it crystal clear for which attack the additional information must be provided. On the other hand, SecBot is also trained to correctly classify such messages that do not contain directly any information in the form of entities about the attack. This is done by including examples such as the ones from line 3 to 6. The other remaining intents of this feature, such as *request_further_attack_classification* or *request_attack_comparison*, are defined in the same way. Since the parameter *use_masked_language_model* of the *DIETClassifier* is set to *True*, SecBot is able to distinguish between the subtle differences of the individual intents.

```
1 - intent: request_attack_information
2   examples: |
3     - Alright, can you elaborate what this attack does?
4     - what does this attack do?
5     - Can you explain what this attack does
6     - How does it work?
7     [...]
8     - What is a [Phishing scam](attack_name)?
```

```
    - Can you explain what a [dDOS](attack_name) attack is?
    - Explain [DNS amplification](attack_type) attacks
    - How does [ransomware](attack_name) work?
    [...]
```

Listing 6.12: Intent - Request Attack Information

Moreover, for each of these intents, a rule is created which directly executes the corresponding custom action. So does, for instance, the *request_attack_comparison* intent always invoke the *action_provide_attack_comparison* action. I favored rules over stories to handle the execution of those custom actions, as the same questions should always output the same answers. And with rules, the probability of predicting the wrong action after correctly classifying the intent is significantly reduced.

All custom actions of the attack support feature are similarly structured, but tailored to their own purpose. Once such an action is executed it

1. retrieves all attacks that are supported by SecBot,
2. retrieves all attacks for which certain information is asked about,
3. checks whether the requested attacks are valid, i.e. supported,
4. outputs the requested information,
5. and stores the explained attack(s) in the corresponding slot.

As mentioned earlier, storing the information within the NLU training data would be beyond the numbers of intents and response actions. That's why I came up with the idea of outsourcing these data into a JSON file. Listing 6.13 illustrates the *attack_information.json* file, which is mainly characterized through nested dictionaries. The keys of the outer dictionary constitute the supported major attacks which contain further dictionaries as values. The first nested dictionary contains five more keys, where four of them (such as lines 3, 17, 18, 19) provide specific information regarding the major attack. The fifth key *subtypes* on line 4 has as its value another dictionary in which all subtypes of the major attack are again listed as key - value pairs, with the value containing the specifications of the subtype. The keys *characteristics*, *scope* and *challenges* thereby refer to the specific tables in Chapter 2.1, where selected subtypes of the respective major attack were presented. The *specific_attacks* key is an optional key for the subtypes of the major attacks. If this key is available, then all the specific attacks of a subtype are briefly described in there. For example, the subtype *protocol-based DDoS* would include *SYN flood* as a specific attack.

```
1 {
2     "ddos": {
3         "characteristics": "attack description",
4         "subtypes": {
5             "subtype_1": {
6                 "characteristics": "...",
7                 "scope": "...",
8                 "challenges": "...",
```

```
                "specific_attacks": {
                    "specific_attack_1": "description"
                }
            },
            "subtype_2": {...},
                .
                .
        },
        "challenges": "...",
        "impacts": ["list including all effects DDoS may cause"],
        "countermeasures": ["list including general protection"]
    },
    "malware": {...},
    "phishing": {...}
}
```

Listing 6.13: Attack Information JSON

Indeed, all attacks must adhere to the structure just presented above. These include the major attacks *malware* and *phishing*, but also applies to newly introduced cyberattacks. Otherwise, the designed custom actions would not work.

When it comes to retrieving all supported attacks the respective actions execute a private method called *__get_supported_attacks(self)* which returns a tuple including a list and a dictionary. In a first step, the method loads the *attack_information.json* file. Afterwards, multiple loops through the nested dictionaries are started. During the iterations, all attack names, including the major attacks, subtype attacks as well as the specific attacks, are appended to a data structure of type list, which is then stored as the first element of the returned tuple. This list is later used to check whether the extracted attack from the user message is supported by SecBot or not. At the same time, a nested dictionary which describes the relationships between the supported attacks is created. A major attack then has as its value a dictionary with the following keys: *children*, *grandchildren* and *attack_type*. The *children* key would refer to a list that contains all subtypes of the major attack, and the *grandchildren* key has as its value also a list which includes the specific attacks of all its subtypes. Similarly, a subtype would have a *parent* key referring to the major attack and a *children* key referring to its specific attacks. Lastly, each specific attack has also a *parent* key that refers to the subtype it belongs as well as a *grandparent* referring to the actual major attack. Obviously, the subtypes and specific types also contain an *attack_type* key. This key is then used to specify the current attack as either major, subtype, or specific attack. This dictionary is then returned as the second element of the tuple.
The reason for creating such a dictionary is to simplify the complexity of looping through nested dictionaries when it comes to retrieving the required information. Instead of nested loops, we can simply retrieve the required keys from the relationship dictionary and then access the data directly. Listing 6.14 provides an illustrative example of a relationship dictionary.

```
{
    "ddos": {
        "children": ["application layer", "protocol based"],
        "grandchildren": ["http flood", "syn flood"],
```

```
        "attack_type": "major_attack"
    },
    "application layer": {
        "parent": "ddos",
        "children": ["http flood"],
        "attack_type": "subtype_attack"
    },
    "http flood": {
        "grandparent": "ddos",
        "parent": "application layer",
        "attack_type": "specific_attack"
    },
    [...]
}
```

Listing 6.14: Attack Relationship Dictionary

When it comes to retrieving all attacks for which certain information has been requested, another private method is executed. When invoked, this method takes the current *Tracker* instance as an argument. *Tracker* is a class which tracks the conversation between the assistant and the user. It allows to access the stored information, i.e. the bot memory, within the custom actions. In this sense, information about past events, such as which slots have been set most recently or retrieving the last user message, can be easily obtained. As a first step, this method retrieves all entities from the latest user message and stores them as a list, as shown in Listing 6.15 on line 5. If that list is empty as indicated on line 7, then the *Tracker* is used to find out which action was last executed. If the latest executed action is one that belongs to the attack support feature, the *referred_attacks* variable on line 3 is then set to the corresponding slot that had been previously set by that specific action and finally returned. Otherwise, the boolean value *False* will be returned.
If there is at least one element in the list on line 5, the algorithm checks whether these entities are of type *attack_name* or *attack_type*. By fulfilling this requirement the entities will be appended to the *referred_attacks* list and then returned by the method. Otherwise, the boolean value *False* is returned.

```
1  def __get_referred_attacks(self,..., tracker: Tracker):
2  
3      referred_attacks = []
4  
5      latest_entities = tracker.latest_message.get("entities", [])
6  
7      if len(latest_entities) == 0:
8          [...]
9      else:
10         [...]
11 
12     return referred_attacks
```

Listing 6.15: Retrieve Requested Attacks

This is how this method is designed for almost all custom actions of this feature. In the case of the *action_provide_attack_comparison* action, the code sketch in Listing 6.15 must be tailored more to the event of comparing attacks. In order to perform a comparison,

there must be at least two attacks. In this sense, when retrieving the requested attacks, we need to add another scenario for a user message that contains only one attack entity. Then, for the first two cases where not enough attacks were provided for the comparison request, backtracking the conversation stored in the *Tracker* must be performed accordingly before returning either the *referred_attack* list or the *False* value.
Therefore, the ability to trace back the conversation makes this feature more context conscious and sets it above the basic FAQ.

Once all requested attacks are collected, the algorithm checks whether they are supported by SecBot. This is done by checking if the attacks are included in the list that was returned by the *__get_supported_attacks(self)* method. If no attack, or less than two attacks in the case of attack comparison requests, pass the check, SecBot then notifies the user that a more specific message is required, as not enough supported attacks could be extracted. Moreover, this notification is also displayed when the *__get_referred_attacks(...)* method returns *False*.

With the help of the previously described relationship dictionary, the requested information is retrieved within the *attack_information.json* file and finally displayed to the user. In addition, each custom action that belongs to the attack support feature and that has been successfully completed sets a slot, which relates to the action, to the successfully processed attacks (cf. Subchapter 5.2.5). This step then facilitates the retrieval of the required attacks when none could be extracted from the user message.

Another thing worth noting is that when it comes to displaying more than one piece of information, such as when the user asks what symptoms indicate a particular attack, or simply asks for the impacts of an attack, I have implemented two different variations, which the administrator of SecBot then can choose which one to set as default. Therefore, you can either, for instance, display each symptom one after another, or, for the other variant implemented for the *action_provide_attack_impacts* action, first display one possible impact of the attack while noting that there are many more impacts to consider. Subsequently, a link to a file containing all the impacts the requested attack could have is displayed.

Moreover, as we have seen, the attack support feature provides the possibility to interact with SecBot also only via one-turn interactions. For example, one could ask what a DDoS attack is. The next question would then relate back to a topic covered by this feature but would not depend on the current context, such as asking for the difference between phishing and malware, and so on. Such scenarios introduce new entry points for unexpected input during form actions which were previously explained in detail in Subchapter 6.2. However, the implementation of unexpected input through the use of the attack support functionality differs slightly from the one discussed previously. Although I have used rules to handle such cases before (e.g. for the *chitchat* and *out_of_scope* intents), this time more than one rule had to be defined for each intent of the attack support feature to be able to address these kind of scenarios. Listing 6.16 shows how the rules regarding the attack comparison functionality are implemented. In particular, this means that besides the basic rule on line 2, where the corresponding action is always executed when the respective intent is classified, two additional rules had to be created. The first one starting at line 7 applies during the *symptom_form* as specified on line

10. Additionaly, the parameter on line 14 had to be set to *false* in order to be able to react appropriately. The other rule starting at line 16 is implemented the same way, but tailored to the *more_info_form*. Setting the *wait_for_user_input* parameter to *false* is crucial, since we want to return to the form immediately once the user has received its answer. Furthermore, I trained SecBot through stories to ask the users whether they want to continue or cancel the form after such an input.

```
1  ## rules for attack comparison
2  - rule: execute action_provide_attack_comparison
3    steps:
4    - intent: request_attack_comparison
5    - action: action_provide_attack_comparison
6  
7  - rule: don't wait for user input when execution of
8          action_provide_attack_comparison is unexpected input part 1
9    condition:
10     - active_loop: symptom_form
11   steps:
12   - intent: request_attack_comparison
13   - action: action_provide_attack_comparison
14   wait_for_user_input: false
15 
16 - rule: don't wait for user input when execution of
17         action_provide_attack_comparison is unexpected input part 2
18   condition:
19     - active_loop: more_info_form
20   steps:
21   - intent: request_attack_comparison
22   - action: action_provide_attack_comparison
23   wait_for_user_input: false
```

Listing 6.16: Unexpected Input

6.4 Integration with Telegram

In order to make SecBot more user-friendly, a more intuitive user interface is needed instead of Rasa's current CLI. In this chapter, I therefore provide guidance on how to integrate SecBot with the popular messaging platform Telegram.

As a first requirement, you need to have an own Telegram account. After logging into your account you need to reach out to Telegram's BotFather and create a new Telegram bot. For this purpose, you enter the command *"/newbot"* and choose a name and a user name for your bot. Upon successful completion of these tasks, you will receive a token from the BotFather that you can use to access the HTTP API.

Once you have received the bot credentials, you then have to set them in SecBot's *credentials.yml* file. The token you got from the BotFather will be your *access_token* and the username of your Telegram bot will be the *verify* as indicated in Listing 6.17.

```
telegram:
  access_token: "<your access token>"
```

```
  verify: "<bot username>"
  webhook_url: "<your_url.com/webhook>"
```

Listing 6.17: Adjusting Credentials File

Unfortunately, there is no way to set the *webhook_url* directly to the *localhost* endpoint. To use webhooks, the bot server must be reachable from the Internet and, in the case of Telegram, a secure HTTPS connection is required. This means that you either use an existing domain URL of yours or you buy such a domain name and then set the URL accordingly.

Another possibility comes with the use of Ngrok. By setting up a Ngrok tunnel a new HTTPS URL is created for free and connects your *localhost* to the Internet. This allows you to test your assistant online with a real life channel. The only drawback is that whenever you close the Ngrok tunnel and restart it, a new URL is generated, which then has to be configured again with the assistant. There is also a way to obtain a permanent URL, but it costs just like the previously mentioned solutions.

To set up a Ngrok tunnel is simple. You first have to create an account in order to be able to download the software. It comes in a single executable file which you have to open. The Rasa server runs on the *localhost:5005*. Therefore, you have to create an URL for port 5005. To do this execute the following command: *ngrok http 5005*.

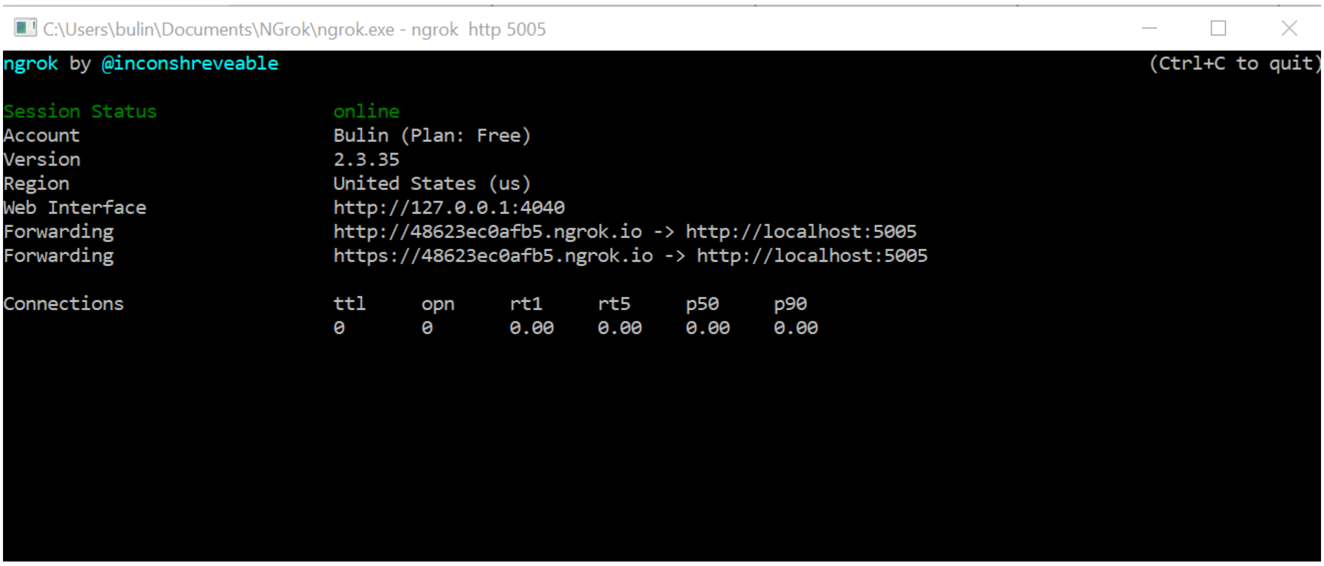

Figure 6.1: Create secure HTTPS endpoint

The Ngrok tunnel automatically creates both a HTTP and a HTTPS endpoint, as demonstrated in Figure 6.1, which are then both forwarded to the *localhost:5005*. The newly created HTTPS connection can then be assigned to the *webhook_url*. Moreover, as shown in Listing 6.18 on line 1, */webhooks/telegram/webhook* must also be appended to the newly created URL. Next, the webhook for the Telegram bot must be set. This is done by running the command on line 3 in Listing 6.18 on a webbrowser of your choice, where *my_bot_token* refers to to your bot's access token and the *url_to_send_updates_to* being the HTTPS link Ngrok provided you with.

```
webhook_url: "https://48623ec0afb5.ngrok.io/webhooks/telegram/webhook"

https://api.telegram.org/bot{my_bot_token}/setWebhook?url={
    url_to_send_updates_to}
```

Listing 6.18: Setting up Webhook

Once you have completed the configuration, you can start the Rasa and the action server by typing the following commands in the terminal: *rasa run* and *rasa run actions*. After that, SecBot should be up and running on Telegram.

Chapter 7

Evaluation

This chapter first provides a case study that simulates a hypothetical real-world situation. The second part then focuses on SecBot's performance based on different configurations, while the third part includes a discussion and limitations of SecBot. The evaluation was conducted on an HP Pavilion x360 Convertible laptop, which has an Intel Core i5-7200 CPU and 16 GB of RAM.

7.1 Case Study

For this case study, we assume that there is a website called *www.otc-premiumfashion.com* where *otc* stands for *on-the-cheap*. It belongs to a small company that buys quality designer clothes from various verified cross-border and overseas outlets on the cheap, and then retails them on its website with prices that are lower than those of any fashion competitor. The business owner invested a lot in the website to attract as many customers as possible and after a year the company was finally able to establish itself in this market. However, investments in security have been severely neglected.
Suddenly, one day, many customers contacted customer support, with the vast majority stating that they were unable to access the website. Those who could access it reported that the website was very slow to respond. Right away the software engineer who was in charge of the website was informed about the situation.
The software engineer was very skilled when it came to creating exquisite websites, but even less so when it came to network monitoring and Internet security. Nevertheless, during troubleshooting, the software engineer was able to make the following findings:

1. There have indeed been many request made to the server.
2. However, the individual requests showed no signs of transmitting excessively large packets that would eventually saturate the available bandwidth.
3. There have been many different IPs involved in the requests.
4. The overwhelming majority of requests were connection requests (SYN requests), as the TCP connection table was used almost to its maximum.

7.1.1 Attack Identification Scenario

The software engineer suspected that there was some malicious work behind the website's behaviour, but was not able the resolve the issues. Then, while browsing the web, he came across SecBot and decided to give it a try. On the official website of the University of Zurich, the company then acquired the software license.

Then, after acquiring the license and downloading the assistant, the software engineer starts talking to SecBot as illustrated in Figure 7.1. The conversation starts with a casual greeting and a request for help. After that, the software engineer first wants to try out SecBot with the symptoms that the customers have submitted. He therefore enters the descriptions sequentially and waits for SecBot's response. Unfortunately, SecBot was not able to identify the potential attack with the given problem description. But the software engineer seems to be optimistic and wants to give SecBot a second chance by providing additional symptoms. To do this, Figure 7.2 shows how he first asks SecBot whether this is possible or not. Since SecBot supports this feature (cf. Subchapter 5.2.2), the software engineer then delivers his findings (1, 3, and 4) all at once and specifies the server as the new target. However, SecBot would also be able to process even larger and more complex user utterances, mainly due to the current settings of the DIETClassifier (cf. Subchapter 6.1). After the submisson of the addtional symptoms, SecBot has been able to successfully identify SYN Flood as the underlying attack causing the website availability issues. Note that if the identification failed again, the software engineer could then again provide additional symptoms to SecBot or simply start the process from the very beginning (cf. Subchapter 6.2).

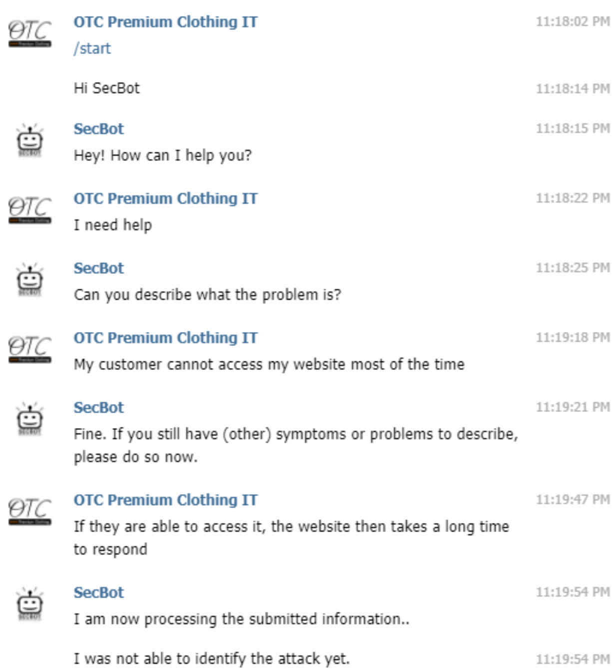

Figure 7.1: Failed Attack Identification

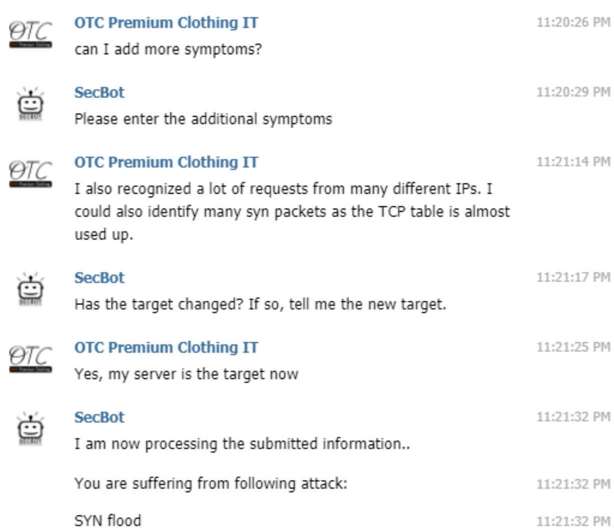

Figure 7.2: Attack Identification

7.1.2 Attack Information Scenario

Now that the software engineer knows from what attack the website is being affected, he now is interested in learning more about that attack. For this purpose, the attack support feature allows him to gain insight into various aspects of an attack (cf. Subchapter 5.2). In order to understand the more detailed information about the attack, it must be clear first what the attack is about in the first place. And since the software engineer does not know anything about the attack's approach, he therefore decides to first ask SecBot to further explain the modus operandi of a SYN flood attack. This part of the conversation is captured in Figure 7.3. You can see that the assistant responded with a thorough explanation which clarifies the basic mechanism of the underlining attack. For the software engineer, this answer was equally crystal clear, so that he now understands why the company's server resources have become so severely depleted, thus leading to low computing performance or even inaccessibility.

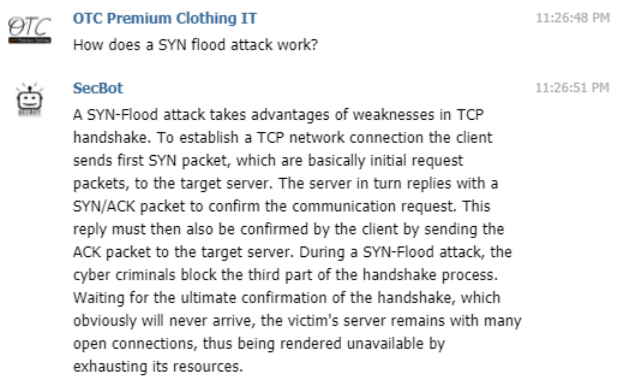

Figure 7.3: Attack Explanation

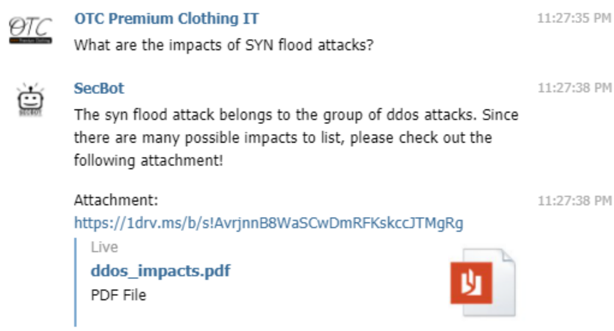

Figure 7.4: Attack Impacts

Having acquired the basic knowledge, the software engineer is now keen to know what other effects the SYN flood attack might have, apart from the fact that the website is no longer accessible as the company is still suffering from the ongoing attack. For that, SecBot first notes that SYN flood belongs to the group of Distributed Denial-of-Service attacks (cf. Subchapter 2.1). Then, as illustrated in Figure 7.4, we can see that the University of Zurich chose the variant of providing a link as an attachment, which then redirects the user to the actual document containing all the necessary information about the impacts of DDoS attacks. The different variations of how to provide more than one piece of information were discussed in Subchapter 6.3 in more detail.

Up to this point, the software engineer has been able to gather some important information. But before finally asking the conversational agent how to mitigate this cyberthreat, the software engineer wants to know if there are any other symptoms of this attack besides the ones he has identified by himself and those reported by the customers that he might have missed. This part of the conversation is shown in Figure 7.5. Other than previously illustrated in Figure 7.4, SecBot now displays all the potential symptoms of the main attack one after the other. As you can see, there are also a few problem specifications included, such as Website or service becomes unexpectedly slow or all of a sudden unavailable, which the company's customers have mentioned during their complaints.

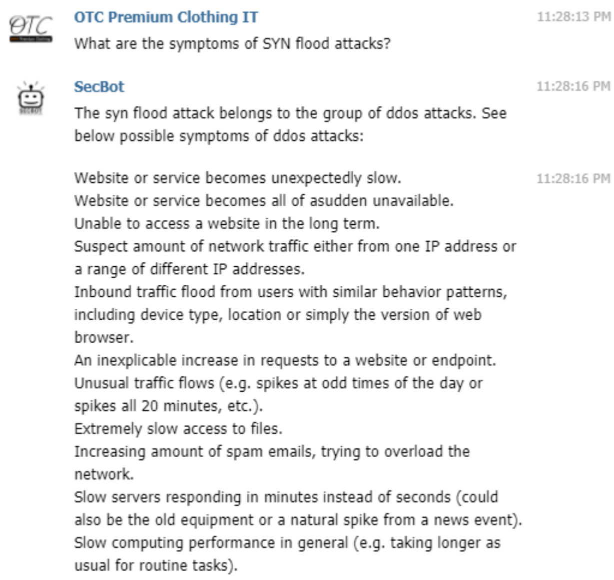

Figure 7.5: Attack Symptoms

7.1.3 Protection Measures Scenario

Now the software engineer has a lot of insightful information about the identified attack. He understands what the company is up against and is now determined to address the issue. However, as mentioned at the beginning of the chapter, the software engineer is not a security expert. Since SecBot has proven itself so far, the software engineer decides to ask the assistant again and is now interested in knowing what countermeasures are available to mitigate such attacks. As shown in Figure 7.6, SecBot presents a few solutions which are based on the underlining main attack.

In a next step, the software engineer might ask for a suitable *Web Application Firewall* for the *www.otc-premiumclothing* website. In this sense, SecBot first extracts all the necessary information about the company, budget, industry, etc. Then, it connects to a recommender system that provides suitable protection solutions for services. Preferably, the recommender system will be a homegrown product of UZH, such as the already existing recommender system MENTOR [22]. This would allow effective attack support from start to finish. Unfortunately, this last part could not be implemented due to time constraints.

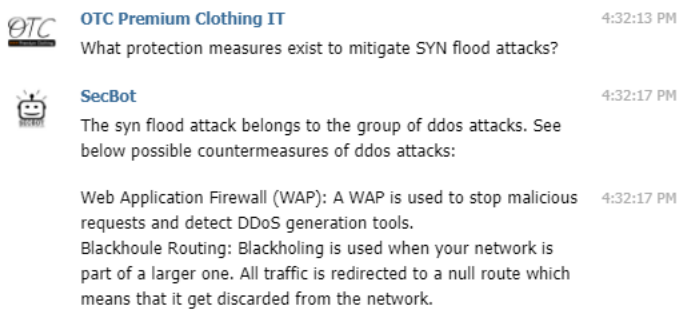

Figure 7.6: Attack Countermeasures

7.2 Performance

Since the start of the thesis I have constantly adjusted SecBot's configuration file. In order to achieve a higher accuracy for the model, not only the training data had to be optimized, but also the pipeline and the policies section within the *config.yml* file. Especially the pipeline has experienced a lot of change. For this reason I will discuss the performance of three pipeline settings regarding intent classification and entity recognition in order to show whether this subgoal has been fulfilled.

For this purpose I have tested different pipeline configurations where the main differences are illustrated in Table 7.1. The complete pipelines are included in Appendix D.
To conduct the tests, I have run all the benchmarks on Rasa's CLI using the following command in Listing 7.1:

```
rasa test nlu --config config.yml --cross-validation --runs 1 --folds 2
   --out results/config
```

Listing 7.1: Run Benchmarks using CLI

The - - *config* flag allows to specify the configuration file, in this case *config.yml*, which should be tested. By setting the - - *cross-validation* flag the test will run on different sets of data. In particular, the - - *folds 2* flag signifies that the existing nlu data will be split into a training set and a validation set. By default, Rasa performs the testing three times for each specified configuration. This actually consumes a lot of hardware resources. Due to hardware restrictions, I have run these tests only once, as specified with the - - *runs* flag.

Table 7.1: Configurations to test

Configuration	**Main Differences**
initial	Uses *SklearnIntentClassifier* for intent classification and *CRFEntityExtractor* for entity recognition
diet	Uses *DIETClassifier* for both intent classification and for entity recognition
diet-masking	Hyperparameter *use_masked_language_model* of *DIETClassifier* is set to *true* and *epochs* to 400

The *rasa test* command generates various reports and charts which can be further used for an in-depth analysis of the respective model. For the purpose of comparing different pipelines, the subsequent diagrams show the average values for the following metrics:

- **Precision:** Also known as specificity, precision is defined as $\frac{TruePositive}{TruePositive+FalsePositives}$. More generally, precision indicates how accurate the model is, i.e., it indicates how many of the predictions are actually correct (percentage of relevant results). This metric is usually used when the cost of an incorrect prediction is high [56].

- **Recall:** Also known as sensitivity, recall is defined as $\frac{TruePositive}{TruePositive+FalseNegative}$. Put in more general terms, it represents the percentage of relevant results that are correctly classified by the model. This metric is commonly used when the costs of a missed prediction is high [56].

- **F1-Score:** The f1-score is an average of the precision and the recall and is defined as $2 \times (\frac{Precision \times Recall}{Precision+Recall})$. This metric is needed if you want to achieve a balance between precision and recall [56].

So let's first get an overview of how well our dialogue management model performed in terms of intent classification under the configurations shown in Table 7.1.

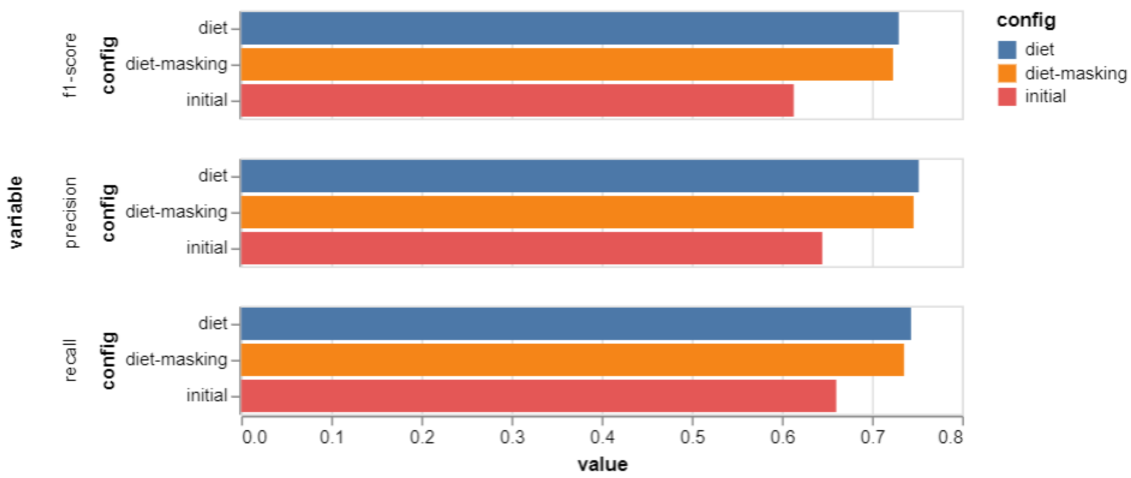

Figure 7.7: Intent Summary Overview

At first sight, it seems that the f1, precision and recall scores in Figure 7.7 all have the same chart. The *initial* configuration depicts the actual configuration before the start of the thesis and is represented by the blue bar. We quickly see that it has the worst performance in all three categories compared to the other two configurations. Furthermore, the *diet* configuration appears as a blue bar within the figure, while the orange bar represents the *diet-masking* configuration.

To be able to directly compare the *diet* with the *initial* configuration, I have therefore used the default settings for all customizable hyperparameters for both pipelines. As you can see in Figure 7.7, the *diet* configuration performs best in all three areas. Regarding recall, SecBot achieves an average value of 74.32% when using the *diet* pipeline, thus obtaining a performance boost of 8.3% compared to the original configuration (66.02%). For precision, an average value of 75.18% is obtained, corresponding to an increase of up to 10.7% over the *initial* pipeline (64.48%). Finally, the f1-score for the *diet* configuration also outperforms the *initial* configuration (61.31%) with an average value of 72.96%, boosting SecBot's performance by 11.7%. The results for the *diet-masking* configuration

are only marginally lower than those of the *diet* configuration with 74.61% for the precision, 73.55% for the recall and 72.33% for the f1-score. One reason that configuration using masking underperforms could be that this configuration needs more epochs to pass by, or on the contrary, as we suspected, that the data set in question is not necessary a complicated one. However, if we further increase the number of epochs, we then risk an overfitted model that does not generalize well.
Nevertheless, the differences between these two pipelines are marginal. What is more important is that compared to the original pipeline, both configuration achieve a performance increase of around 10%, which is not nothing, but actually quite substantial when it comes to accurately classifying intents. Whether we choose the *diet* configuration or the *diet-masking* configuration now depends on how well they performed in terms of entity recognition.

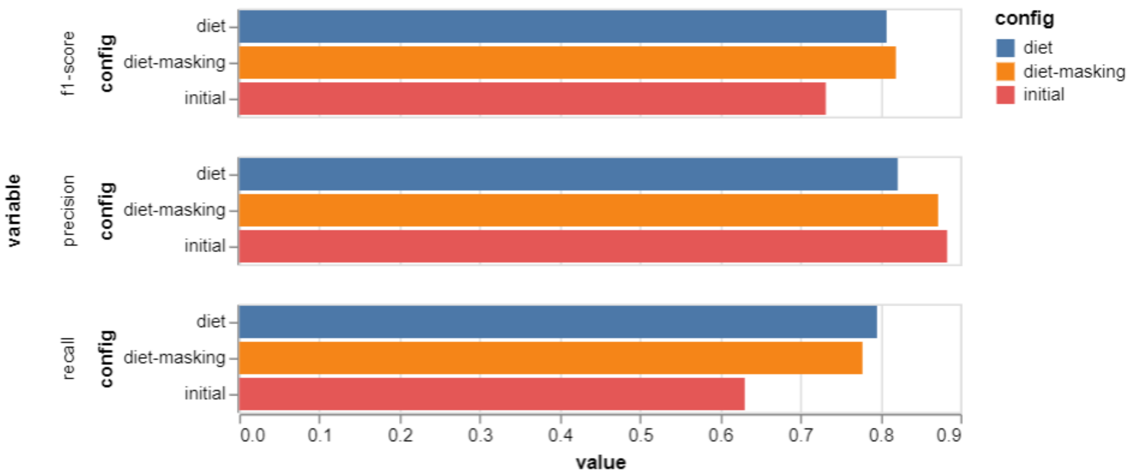

Figure 7.8: Entity Summary Overview

And now let's have a look of how well the model performed in terms of entity recognition using the different configurations. The colors for the different pipelines are the same as in Figure 7.7. Other than before, the charts in Figure 7.8 do not look quite the same as in the previous figure. Inspecting the values for the recall category, we can see that again the *diet* pipeline outperforms the other two configurations with an average score of 79.58%. The performance boost compared to the *initial* configuration thereby amounts to a whopping 16.5%. And again, the *diet-masking* pipeline underperforms only slightly worse than the *diet* configuration with an average value of 77.76% , but still performs substantially better than the original SecBot configuration.
In terms of precision, however, the *initial* configuration performs surprisingly well. Indeed, with an average of 88.33% it marginally outperforms the *diet-masking* configuration (87.20%) by a margin of 1.13%. The *diet* configuration still achieves a high precision value of 82.17%, but performs worst compared to the other two pipelines with a significant performance decrease of 6.16% from the originial pipeline.
Finally, in terms of f1-score, we again have a new top performer. In this case, the *diet-masking* configuration with an average score of 81.93% just barely surpasses the *diet* configuration (80.77%) and achieves a significant performance increase of 8.74% over the *initial* configuration (73.19%).

As mentioned before, would it be only for the intent classification, we would obviously drop the *initial* configuration for one of the other two configurations. However, in terms of entity recognition, the results don't seem so clear-cut. Each area has a different top performer. In such a case, we need to prioritize which metric is most important to us. And in our case, precision is the one we should aim for. In order to achieve high usability the user intents must be correctly identified. Moreover, this step is followed by correctly extracting entities. If entity recognition doesn't work well, then the implemented custom actions might fail and the user will not receive accurate responses. This, in turn, could lead to giving wrong information and proposing protection measures that are not viable, resulting in a significant loss for the customer while the original problem still exist. Therefore, we have to look for the optimal trade-off. In this sense, although the *initial* configuration slightly outperforms the other two regarding precision, we need to favour one of configurations using the *DIETClassifier*, as they relatively offer a massive performance gain in terms of intent classification than the performance increase offered by the original pipeline in terms of entity recognition.
Finally, the configuration chosen from now one is the *diet-masking* pipeline. Once again, although the *diet-masking* configuration has a minimal lower precision value for the intent classification, it offers all the more a relatively higher precision for entity extraction.

To sum up, the changes in both intent classification and entity recognition have given SecBot a massive performance boost. In addition, SecBot now works much better on the recall as well. And finally, since the f1-score is an average of the recall and precision, it shows that SecBot's overall performance now is quite well despite having a slightly lower precision in entity extraction.

7.3 Discussion and Limitations

As we have seen in Subchapter 7.1, the case study deals with various aspects that were implemented during the thesis. It starts with the transmission of all necessary information, continuous by querying further attack details, until common countermeasures have been proposed. However, the complexity encountered during the conversation in attack identification scenario is sophisticated, as the first attempt to identify the attack failed, but still not that high. Nevertheless, due to the configurations made (cf. Subchapter 6.1) and various other features presented in Subchapter 6.2 SecBot is quite capable of handling much more complex situations, where the user may, for example, want to provide larger sentences in general, to specify additional symptoms multiple times or simply restart the form after a failed identification. This also corresponds to the other scenarios presented in Subchapter 7.1.2 and 7.1.3. The respective parts of the conversations are simply illustrated as one-turn interactions, which is meant to simplify the conversation flow so that the reader can easily follow along with the figures. In reality, however, it is possible to have a more contextualized conversation with SecBot (cf. Subchapter 6.3). Furthermore, as seen in the use case scenarios, SecBot was able to always provide accurate and thorough responses, thus always clarifying the user's questions.

Moreover, as indicated in Subchapter 7.1.2, some implementation decisions have to be made before going into production. This is especially the case when more than one piece of information needs to be displayed, such as possible symptoms, impacts or protection measures. I've shown two options, either display the information one after the other or provide a link to a PDF file or even a new webpage where all the requested information is presented. Depending on which channel SecBot is to be integrated into, there may be other possibilities as well.

Another limitation of SecBot is the lack of training data regarding cyberthreats. As of now, SecBot is well trained for three major attacks and chosen subtypes of them and can also provide insighful information. But we all know that this covers only a small fraction of cyberattacks since there exist a plethora of other attacks.

Furthermore, although SecBot has now become a more sophisticated prototype, it is nevertheless very limited when it comes to identifying the possible attack. This is due to another core functionality, namely the attack identification algorithm, which is still underdeveloped and has only few symptoms hard-coded to specific cybersecurity threats.

What's more is that as a rather newer framework, Rasa is in constant state of change. On the one hand, this might be extremely advantageous since new and improved features can greatly enhance your assistant, as the introduction of forms and rules to SecBot has shown. On the other hand, this could constitute a new challenge as well. We have seen it with the introduction of Rasa 2, where all code from Rasa 1 suddenly turned into legacy code. Development after migration wasn't an easy task either due to the lack of detailed documentation and tutorials. Moreover, it even took several more releases to cover unexpected bugs and errors in the new code of Rasa 2. With this in mind, keeping SecBot up to date in this fast-moving environment will not be an easy task. However, this is rather an environmental restriction. We could also implement SecBot with Google's DialogFlow or another more mature AI framework.

Chapter 8

Summary and Conclusions

The overall objective of this thesis was to provide a prototype that allows end-users to input their demands for cybersecurity support, thus, generating an accurate answer which then will be used by the user during the cybersecurity decision-making process. In this sense, besides the refinement of the initial prototype of SecBot, new features and knowledge had to be implemented.

After conveying the general knowledge about selected cybersecurity threats and chatbot development, a brief overview of related work is given, with a particular focus towards conversational agents. Thereafter, SecBot is presented in more detail. In particular, this chapter highlights SecBot's capabilities, while also providing an overview of SecBot's initial implementation. After this first theoretical part of this work, the following chapter first narrows down the scope of the thesis and then introduces the novel functionalities and improvements on a high level. This chapter is subsequently followed by a detailed description of the current implementation of the proposed functionalities and improvements, with an emphasis on the most important parts. Afterwards, the usability of SecBot is evaluated through a case study based on three use case scenarios. In addition, this chapter also provides an overview of SecBot's NLU performance, which is then followed by a discussion and limitations of SecBot.

As of now, SecBot is able to effectively gather all the necessary information from the user in order to possibly identify a potential attack, while assuring that unexpected input will be handled accordingly. Moreover, thorough explanations for attack-specific issues can now be provided, either in the form of one-turn interactions or in a more advanced and contextualized conversation. Lastly, due to the improved configuration, SecBot's accuracy could be significantly increased.

Future work should definitively focus primarily on the optimization of the attack tree algorithm. As it stands today, this algorithm is highly restricted since it has a limited set of symptoms related to some cyberattacks hard-coded. Moreover, expanding the knowledge base is another area to focus on in the future. Therefore, not only the NLU data but also the freshly created JSON file, whose structure allows for easy expansion of the document with additional cyberthreats, should be extended with more examples of cyberattacks. Additionally, as indicated in the case study, one should consider to have a cybersecurity recommender system, such as MENTOR [22], integrated into SecBot. This would not only allow to directly suggest appropriate solutions based on the information provided by the user, but would also reduce the usability overhead of recommender systems as SecBot would act as an intermediary, meaning that users could specify their demands for the requested protection in natural language. Finally, due to the increase in popularity, there is a rising concern that chatbots could become the next gateway for cyberattacks. And since SecBot is all about cybersecurity, secure usability for the user should be ensured. Therefore, there are many techniques, such as end-to-end encryption, user authentication and authorization, self-destructing messages or secure protocols that can be considered.

Bibliography

[1] 2019 data breach investigations report. Technical report, Verizon, 2019.

[2] 2020 user risk report exploring vulnerability and behaviour in a people-centric threat landscape. Technical report, Proofpoint, 2020.

[3] Jenna Alburger. Rule-Based Chatbots vs. AI Chatbots: Key Differences, 2020. https://www.hubtype.com/blog/rule-based-chatbots-vs-ai-chatbots/, last visit November 2020.

[4] Antaviana. The cybersecurity directory to tango with. [Online] https://www.cybertango.io/, last visit October 2020.

[5] Dania Ben-Peretz. A Siri for Network Security: How Chatbots can Enhance Business Agility, 2020. https://www.infosecurity-magazine.com/opinions/network-chatbots-agility, last visit November 2020.

[6] Chris Brook. What is security as a service? a definition of secaas, benefits, examples, and more. December 2018, [Online] https://digitalguardian.com/blog/what-security-service-definition-secaas-benefits-examples-and-more, last visit October 2020.

[7] Cisco. What Is Phishing? https://www.cisco.com/c/en/us/products/security/email-security/what-is-phishing.html?dtid=osscdc000283, last visit December 2020.

[8] Cloudflare. Application Layer DDoS Attack. https://www.cloudflare.com/learning/ddos/application-layer-ddos-attack/, last visit November, 2020.

[9] Cloudflare. HTTP Flood Attack. https://www.cloudflare.com/learning/ddos/http-flood-ddos-attack/, last visit November, 2020.

[10] Cloudflare. SYN Flood Attack. https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/, last visit November, 2020.

[11] Cloudflare. What is a DDoS Attack? https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/, last visit December, 2020.

[12] Cloudflare. What is Malware? https://www.cloudflare.com/learning/ddos/glossary/malware/, last visit November, 2020.

[13] Louis Columbus. Cybersecurity spending to reach $123b in 2020. August 2020, [Online] `https://www.forbes.com/sites/louiscolumbus/2020/08/09/cybersecurity-spending-to-reach-123b-in-2020/`, last visit October 2020.

[14] Comodo Group. What is a Computer Virus and its Types, 2020. `https://antivirus.comodo.com/blog/computer-safety/what-is-virus-and-its-definition/`, last visit November 2020.

[15] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. Supervised learning. In *Machine learning techniques for multimedia*, pages 21–49. Springer, 2008.

[16] Christos Douligeris and Aikaterini Mitrokotsa. Ddos attacks and defense mechanisms: classification and state-of-the-art. *Computer Networks*, 44(5):643 – 666, 2004.

[17] Drift. An Introduction to A.I. Chatbots. `https://www.drift.com/learn/chatbot/ai-chatbots/`, last visit November 2020.

[18] IBM Cloud Education. Natural Language Processing, 2020. `https://www.ibm.com/cloud/learn/natural-language-processing`, last visit December 2020.

[19] Expert System. Chatbot: What is a Chatbot? Why are Chatbots Important?, 2020. `https://expertsystem.com/chatbot/`, last visit November 2020.

[20] Bobby Filar. Artemis: an intelligent assistant for cyber defense, 2017. `https://www.elastic.co/de/blog/artemis-intelligent-assistant-cyber-defense`, last visit November 2020.

[21] Executive Agency for Small, Medium sized Enterprises, Capgemini Invent, Technopolis, and European Digital SME Alliance. *Skills for SMEs: cybersecurity, Internet of things and big data for small and medium sized enterprises.* Publications Office, 2020.

[22] M. Franco, B. Rodrigues, and B. Stiller. MENTOR: The Design and Evaluation of a Protection Services Recommender System. In *15th International Conference on Network and Service Management (CNSM 2019)*, pages 1–7, Halifax, Canada, October 2019. IEEE.

[23] M. F. Franco, B. Rodrigues, E. J. Scheid, A. Jacobs, C. Killer, L. Z. Granville, and B. Stiller. SecBot: a Business-Driven Conversational Agent for Cybersecurity Planning and Management. In *16th International Conference on Network and Service Management (CNSM 2020)*, pages 1–7, Izmir, Turkey, 2020.

[24] Muriel Figueredo Franco. SecBot - GitLab Repository. `https://gitlab.ifi.uzh.ch/franco/secbot`, last visit December 2020.

[25] Muriel Figueredo Franco, Bruno Rodrigues, and Burkhard Stiller. On the recommendation of protection services. Technical Report No. ifi-2019.06, Department of Informatics IfI, University of Zurich UZH, August 2019.

[26] Dr. Michael J. Garbade. A Simple Introduction to Natural Language Processing, 2018. `https://becominghuman.ai/a-simple-introduction-to-natural-language-processing-ea66a1747b32`, last visit December 2020.

[27] Gartner. Forecasts Worldwide Information Security Spending to Exceed $124 Billion in 2019, 2018. [Online] `https://www.gartner.com/en/newsroom/press-releases/2018-08-15-gartner-forecasts-worldwide-information-security-spending-to-exceed-124-billion-in-2019`, last visit October 2020.

[28] A. Girma, M. Garuba, J. Li, and C. Liu. Analysis of ddos attacks and an introduction of a hybrid statistical model to detect ddos attacks on cloud computing environment. In *2015 12th International Conference on Information Technology - New Generations*, pages 212–217, 2015.

[29] The Cyber Helpline. Supporting victims of cybercrime. `https://www.thecyberhelpline.com/`, last visit November 2020.

[30] Hornetsecurity. Computer Worm. What are Computer Worms? Hornetsecurity explains how Computer Worms work. `https://www.hornetsecurity.com/en/knowledge-base/computer-worm/`, last visit November 2020.

[31] Imperva. What is a dns amplification attack. [Online] `https://www.imperva.com/learn/ddos/dns-amplification/`, last visit November 2020.

[32] Arthur Jacobs, Ricardo Pfitscher, Rafael Ribeiro, Ronaldo Ferreira, Lisandro Granville, and Sanjay Rao. Deploying natural language intents with lumi. pages 82–84, 08 2019.

[33] Eda Kavlakoglu. NLP vs. NLU vs. NLG: the differences between three natural language processing concepts, 2020. `https://www.ibm.com/blogs/watson/2020/11/nlp-vs-nlu-vs-nlg-the-differences-between-three-natural-language-processing-concepts/`, last visit December 2020.

[34] Deepak Kumar, Narender Kumar, and Aditya Kumar. Computer viruses and challenges for anti-virus industry. *International Journal Of Engineering And Computer Science*, pages 3869–3873, 2014.

[35] Lachlan Mackinnon, Liz Bacon, Diane Gan, George Loukas, David Chadwick, and Dimitrios Frangiskatos. *Cyber Security Countermeasures to Combat Cyber Terrorism*, pages 234–257. 03 2013.

[36] Malwarebytes. Computer Virus – Virus Protection & Removal. `https://www.malwarebytes.com/computer-virus/`, last visit November, 2020.

[37] Malwarebytes. Malware. `https://www.malwarebytes.com/malware/`, last visit November, 2020.

[38] Malwarebytes. Ransomware. `https://www.malwarebytes.com/ransomware/`, last visit November, 2020.

[39] Mady Mantha. Introducing DIET: state-of-the-art architecture that outperforms fine-tuning BERT and is 6X faster to train, 2020. `https://blog.rasa.com/introducing-dual-intent-and-entity-transformer-diet-state-of-the-art-performance-on-a-lightweight-architecture/`, last visit January 2021.

[40] Nate Lord. What is Ransomware as a Service? Learn About the New Business Model in Cybercrime, 2020. https://digitalguardian.com/blog/what-ransomware-service-learn-about-new-business-model-cybercrime#:~:text=Ransomware\%20is\%20a\%20type\%20of,phishing\%20emails\%20and\%20infected\%20websites., last visit November 2020.

[41] Netscout. HTTP Flood DDoS Attacks. https://www.netscout.com/what-is-ddos/http-flood-attacks, last visit November, 2020.

[42] Oracle. What Is a Chatbot? https://www.oracle.com/chatbots/what-is-a-chatbot/, last visit November 2020.

[43] Snigdha Patel. What is Chatbot? Why are Chatbots Important?, 2020. https://www.revechat.com/blog/what-is-a-chatbot/, last visit November 2020.

[44] Jeff Petters. What is an it security audit? the basics. March 2020, [Online] https://www.varonis.com/blog/security-audit/, last visit November 2020.

[45] Casey Phillips. The 3 Types of Chatbots & How to Determine the Right One for Your Needs, 2018. https://chatbotsmagazine.com/the-3-types-of-chatbots-how-to-determine-the-right-one-for-your-needs-a4df8c69ec4c, last visit November 2020.

[46] Rapid7. Phishing Attacks: A Deep Dive with Prevention Tips. https://www.rapid7.com/de/cybersecurity-grundlagen/phishing-attacks/, last visit December 2020.

[47] Rapid7. Spear Phishing Attacks. https://www.rapid7.com/de/cybersecurity-grundlagen/spear-phishing-attacks/, last visit December 2020.

[48] Rapid7. Whaling Phishing Attacks. https://www.rapid7.com/de/cybersecurity-grundlagen/whaling-phishing-attacks/, last visit December 2020.

[49] Rasa. Build contextual assistants that really help customers. https://rasa.com/, last visit December 2020.

[50] Juniper Research. Chatbot Conversations to deliver $8 billion in Cost savings by 2022, 2017. https://www.juniperresearch.com/new-trending/analystxpress/july-2017/chatbot-conversations-to-deliver-8bn-cost-saving, last visit November 2020.

[51] Margaret Rouse, Crystal Bedell, and Peter Loshin. computer worm. https://searchsecurity.techtarget.com/definition/worm, last visit November 2020.

[52] Margaret Rouse, Ben Lutkevich, and Robert Richardson. ransomware, 2020. https://searchsecurity.techtarget.com/definition/ransomware, last visit November 2020.

[53] Corero Network Security. How DDoS Attacks Impact Businesses Across Industries. https://www.corero.com/blog/how-ddos-attacks-impact-businesses-across-industries/, last visit December 2020.

[54] Penta Security. Types of ddos attacks: General breakdown. July 2020, [Online] `https://www.pentasecurity.com/blog/ddos-attacks-types-explanation/`, last visit November 2020.

[55] K. Shah, A. Salunke, S. Dongare, and K. Antala. Recommender systems: An overview of different approaches to recommendations. In *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pages 1–4, 2017.

[56] Koo Ping Shung. Accuracy, Precision, Recall or F1?, 2018. `https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9`, last visit February 2021.

[57] Rob Sobers. 110 must-know cybersecurity statistics for 2020. September 2020, [Online] `https://www.varonis.com/blog/cybersecurity-statistics/`, last visit October 2020.

[58] SoftwareLab. What is a Computer Worm? `https://softwarelab.org/what-is-a-computer-worm/`, last visit November 2020.

[59] Artificial Solutions. Chatbots: The definitive guide (2020). [Online] `https://www.artificial-solutions.com/chatbots`, last visit October 2020.

[60] Steve Morgan. Global Ransomware Damage Costs Predicted To Reach $20 Billion (USD) By 2021, 2019. `https://cybersecurityventures.com/global-ransomware-damage-costs-predicted-to-reach-20-billion-usd-by-2021/`, last visit November 2020.

[61] N. T. Thomas. An e-business chatbot using aiml and lsa. In *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2740–2742, 2016.

[62] Jan Tissler. Phishing, Spear Phishing, Whaling & more: The attacks are becoming more sophisticated, 2020. `https://www.dswiss.com/en/news/phishing-spearfishing-und-more-sophisticated-attacks`, last visit December 2020.

[63] Jessica Twentyman. The Cyber Helpline enlists chatbot in battle to help cybercrime victims, 2019. `https://diginomica.com/cyber-helpline-enlists-chatbot-battle-help-cybercrime-victims`, last visit November 2020.

[64] Estelle Verani. How to Build your Own Transactional Chatbot, 2020. `https://www.inbenta.com/en/blog/transactional-chatbot/#:~:text=What\%20is\%20a\%20transactional\%20chatbot,to\%20accomplish\%20a\%20specific\%20action.`, last visit November 2020.

[65] Suzanne Widup, Dave Hylender, Gabriel Bassett, Philippe Langlois, and Alex Pinto. 2020 verizon data breach investigations report. Technical report, Verizon, 05 2020.

Abbreviations

ACK	Acknowledgement (TCP 3-way handshake)
AI	Artificial Intelligence
AIML	Artificial Intelligence Markup Language
API	Application Programming Interface
bps	Bits per seconds
CEO	Chief Executive Officer
CFO	Chief Financial Officer
CIO	Chief Information Officer
CLI	Command Line Interface
CPU	Central Processing Unit
CSG	Communication Systems Group
DoS	Denial-of-Service
DDoS	Distributed Denial-of-Service
DNS	Domain Name System
FBI	Federal Bureau of Investigation
HTTP	Hypertext Transfer Protocol
IBN	Intent-based networking
ICMP	Internet Control Message Protocol
IP	Internet Protocol
ML	Machine Learning
NLP	Natural Language Processing
NLU	Natural Language Understanding
OSI	Open Systems Interconnection
PoC	Proof of Concept
pps	Packets per seconds
PUP	Potentially Unwanted Program
RaaS	Ransomware as a Service
RAM	Random-Access Memory
rps	Requests per seconds
SDK	Software Development Kit
SME	Small and medium sized enterprise
SYN	Synchronize (TCP 3-way handshake)
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WAF	Web Application Firewall

List of Figures

List of Tables

Listings

Appendix A

Installation Guidelines

This installation guideline is based on a Windows operating system. Therefore, the setup for MacOS or Linux might differ. The source code of SecBot is publicly accessible on the GitLab of the Institute of Informatics of the University of Zurich.

1. **Initial Setup**
   (a) Install Python version 3.7.5
   (b) Install Rasa with `pip3 install rasa==2.2.2`
   (c) Install dependencies with `pip3 install -r requirements.txt`
   (d) Install spaCy dependencies `pip3 install rasa[spacy]`
2. **Clone GitLab Repository**
   (a) Clone the GitLab repository by running
   `git clone https://gitlab.ifi.uzh.ch/franco/secbot.git`
   in the command prompt
3. **Starting SecBot**
   (a) Using the Command Line Interface
      (i) Open the command prompt and direct to the directory where the source code is located
      (ii) Start the action server with `rasa run actions`
      (iii) Start SecBot with `rasa shell` or `rasa shell --debug`
   (b) Using Telegram
      (i) View Subchapter 6.4

Appendix B

Contents of the CD

- Final thesis as PDF
- Intermediate presentation as PDF
- Complete source code as ZIP

Appendix C

SecBot Responses

```
responses:
  utter_welcome:
  - text: Hi, I'm SecBot! How can I assist you?
  utter_ask_protection:
  - text: What kind of protection do you need?
  utter_attack_details:
  - text: Fine. Sarching for protection against {attack_type} {attack_name}
  utter_attack_notif_name:
  - text: Ok. Do you have more information about the {attack_name} attack?
  utter_attack_notification:
  - text: Can you provide more details about the attack?
  utter_attack_pattern:
  - text: Fine. I am processing related problems...
  utter_budget_available:
  - text: What is your budget available to expend in such a protection?
  utter_general_confirmation:
  - text: Ok. I got it.
  - text: Right. I was able to process it
  utter_goodbye:
  - text: Bye
  utter_greet:
  - text: Hey! How can I help you?
  utter_ask_continue:
  - text: Do you want to continue?
  utter_problem_desc:
  - text: Can you describe the problem?
  utter_problem_desc_required:
  - text: You must give me more details or information about what is happening to be able to help you.
  utter_attack_not_identified:
  - text: I was not able to identify the attack yet.
  utter_attack_identified:
  - text: 'I was able to identify the following attack:'
  - text: 'You are suffering from following attack:'
  - text: 'Could identify the attack:'
  utter_target_info:
  - text: Do you know what is the attack's target?
```

```
- text: Do you have information which is the attack's target?
- text: I was not able to understand which is the attack's target.
utter_ask_info:
- text: Do you have this information?
utter_ask_support:
- text: Yes, I can support you with this configuration.
utter_add_symptom:
- text: Do you have more symptoms or problems to describe?
utter_more_symptom:
- text: Sure. Give me more informartion then.
utter_amount_data:
- text: How much critical data do you have to protect?
utter_price:
- text: Do you know the price for this service?
utter_ask_problem:
- text: Can you describe what the problem is?
utter_affirm:
- text: Alright then.
utter_default:
- text: Sorry I didn't get that. Can you rephrase?
utter_please_rephrase:
- text: I'm sorry, I didn't quite understand that. Could you rephrase?
utter_explain_symptoms_1:
- text: You must provide at least one symptom or a brief problem
  description to be able to help you!
utter_explain_symptoms_2:
- text: Adding more information increases the probability of
  identifying the problem / attack
utter_explain_symptoms_target:
- text: I cannot help if I don't know what is being targeted!
utter_explain_more_info:
- text: More information is needed to be able to help you.
utter_explain_new_target:
- text: I need to be sure that the target has not been changed.
  Otherwise I have to adjust it accordingly to identify the correct
  problem / attack.
utter_ask_symptoms_target:
- text: Do you know what is the attack's target?
- text: Do you have information which is the attack's target?
- text: I was not able to understand which is the attack's target.
utter_ask_symptoms_1:
- text: Can you describe what the problem is?
utter_ask_symptoms_2:
- text: Fine. If you still have (other) symptoms or problems to
  describe, please do so now.
utter_continue_symptom_description:
- text: Are you able to provide me with necessary information?
utter_stop_form:
- text: If you want to stop this form then write 'stop' and then
  confirm with 'no'
utter_successfully_stopped_form:
- text: Successfully stopped the form. How can I help you now?
utter_no_help_possible:
- text: In this case, I am can not help you.
utter_ask_more_info:
- text: Please enter the additional symptoms
```

```
utter_ask_new_target:
- text: Has the target changed? If so, tell me the new target.
utter_chitchat/ask_mood:
- text: I'm fine, thanks! How can I help you?
- text: I'm doing great, thanks! How can I help you?
- text: I feel good, thanks! How can I help you?
utter_chitchat/ask_howbuilt:
- text: I was built with a lot of love and patience.
- text: Well, when two chatbots love each other very much...
- text: They always ask how I was built, but never how I am...
- text: I was made by software engineers, but hard work is what built me.
utter_chitchat/ask_howold:
- text: Old enough to be a bot!
- text: Age is just an issue of mind over matter. If you do not mind, it does not matter.
- text: My first git commit was many moons ago.
utter_chitchat/ask_isbot:
- text: Yes, I'm a bot.
- text: Yep, you guessed it, I'm a bot!
utter_chitchat/ask_ishuman:
- text: I'm not a human, I'm a bot!
utter_chitchat/ask_languagesbot:
- text: I can spell baguette in French, but unfortunately English is the only language I can answer you in.
- text: Binary code and the language of love. And English.
- text: I was written in Python, but for your convenience I'll translate to English.
utter_chitchat/ask_time:
- text: It's the most wonderful time of the year!
- text: It's party time!
- text: Time is a human construct, you'll have to tell me.
- text: That's hard to say -- it's different all over the world!
utter_chitchat/ask_whatismyname:
- text: It's probably the one that your parents chose for you.
- text: I'd tell you, but there's restricted access to that chunk of memory.
utter_chitchat/ask_whatisrasa:
- text: Rasa Open Source is a conversational AI framework for that provides machine learning tools for building contextual assistants.
utter_out_of_scope/other:
- text: I can't help you with that, I'm sorry.
utter_out_of_scope/non_english:
- text: I only understand English, I'm sorry.
```

Appendix D

Tested Pipelines

```
1 ## initial configuration before the start of the thesis.
2 ## Line 10 to 18 had to be included so that the assistant still
3 ## worked with the current implementation
4 pipeline:
5 - name: "SpacyNLP"
6 - name: "SpacyTokenizer"
7 - name: "SpacyFeaturizer"
8 - name: "RegexFeaturizer"
9 - name: "CRFEntityExtractor"
10 - name: "EntitySynonymMapper"
11 - name: "SklearnIntentClassifier"
12 - name: ResponseSelector
13   epochs: 10
14   retrieval_intent: chitchat
15 - name: ResponseSelector
16   epochs: 10
17   retrieval_intent: out_of_scope
18 - name: FallbackClassifier
19   threshold: 0.3
```

```
1 ## diet configuration before the start of the thesis.
2 pipeline:
3 - name: "SpacyNLP"
4 - name: "SpacyTokenizer"
5   intent_tokenization_flag: True
6   intent_split_symbol: "+"
7 - name: "SpacyFeaturizer"
8 - name: "RegexFeaturizer"
9 - name: LexicalSyntacticFeaturizer
10 - name: CountVectorsFeaturizer
11 - name: CountVectorsFeaturizer
12   analyzer: "char_wb"
13   min_ngram: 1
14   max_ngram: 4
15 - name: "DIETClassifier"
16   epochs: 300
17 - name: "EntitySynonymMapper"
18 - name: ResponseSelector
19   epochs: 10
```

```
  retrieval_intent: chitchat
- name: ResponseSelector
  epochs: 10
  retrieval_intent: out_of_scope
- name: FallbackClassifier
  threshold: 0.3
```

```
## diet-masking configuration before the start of the thesis.
pipeline:
- name: "SpacyNLP"
- name: "SpacyTokenizer"
  intent_tokenization_flag: True
  intent_split_symbol: "+"
- name: "SpacyFeaturizer"
- name: "RegexFeaturizer"
- name: LexicalSyntacticFeaturizer
- name: CountVectorsFeaturizer
- name: CountVectorsFeaturizer
  analyzer: "char_wb"
  min_ngram: 1
  max_ngram: 4
- name: "DIETClassifier"
  epochs: 400
  use_masked_language_model: True
- name: "EntitySynonymMapper"
- name: ResponseSelector
  epochs: 10
  retrieval_intent: chitchat
- name: ResponseSelector
  epochs: 10
  retrieval_intent: out_of_scope
- name: FallbackClassifier
  threshold: 0.3
```