MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

"Automatic Rule Extraction from Vulnerability Databases for Threat Analysis"

verfasst von / submitted by

Sebastian Chlup, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2020 / Vienna 2020

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 921
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Informatik
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. Dr. Gerald Quirchmayr

Automatic Rule Extraction from Vulnerability Databases for Threat Analysis

Sebastian Chlup
sebastian.chlup@ait.ac.at

Universität Wien - Faculty of Computer Science
Währinger Str. 29, 1090 Wien, Austria

Abstract. The aim of this master thesis is to leverage the capabilities of threat modelling and the threat analysis based on it by enabling a threat model to access up-to-date threat and vulnerability information. This information can come from arbitrary sources in structured or unstructured form. Therefore, this work will discuss an information extraction process and explore a custom model utilized for the representation of the resulting data. The repositories serving as information sources are the National Vulnerability Database and Packetstorm. While the focus in Packetstorm will be put on the extraction of relevant information, the NVD is more sophisticated and contains supplementary attributes that can improve risk treatment. Consequently, suitable mappings of these additional facets will be considered. This thesis shows a way for extending an existing rule-based threat model by automating the process of creating rules from real-world vulnerability data and discusses the development of a prototype.

Abstract. Ziel dieser Masterarbeit ist es, einem existierenden Bedrohungsmodell Zugriff auf neue Bedrohungs- und Verwundbarkeitsinformationen zu ermöglichen und dadurch das Potenzial der Bedrohungsmodellierung und der darauf aufbauenden Bedrohungsanalyse zu steigern. Diese Informationen können aus verschiedenen Quellen stammen und sowohl in strukturierter, als auch unstrukturierter Form vorliegen. Daher wird im Rahmen dieser Arbeit ein Informationsextraktionsprozess eingeführt und ein speziell darauf ausgelegtes Datenmodell diskutiert. Als Informationsquellen dienen sowohl die National Vulnerability Database, als auch Packetstorm. Während der Fokus bezüglich Packetstorm auf die Extraktion relevanter Daten aus unstrukturiertem Text gelegt wird, stellt die NVD einen großteil ihrer Daten strukturiert zur Verfügung. Sie beinhaltet zusätzliche Metriken, welche einen wichtigen Beitrag zum Prozess des Risikomanagements leisten. Folglich wird auch auf mögliche Mappings dieser zusätzlichen Attribute eingegangen. Diese Masterarbeit wird ein existierendes, regelbasiertes Bedrohungsmodell mit Hilfe eines automatisierten Regelerstellungsprozesses um reale Schwachstellendaten erweitern. Des Weiteren wird die Entwicklung eines Prototypen erörtert der die genannten Punkte umsetzt.

Keywords: Threat Modelling · Rule Extraction · Vulnerability · Security · Threat Intelligence · Threat Analysis · Named Entity Recognition

Acknowledgement

First of all, I would like to express my honest appreciation and gratitude to my supervisor Univ.-Prof. Dipl.-Ing. Dr. Dr. Gerald Quirchmayr of the Faculty of Computer Science at the University of Vienna who provided most valuable advice, helpful expertise and patient support during the entire period of writing this master thesis. Regular advisory meetings gave me the feeling to be in good, guiding hands. It is particularly worth mentioning that Univ.-Prof. Dipl.-Ing. Dr. Dr. Gerald Quirchmayr was available even via telephone at any time and always had an open ear for urgent questions.

I am also grateful to the Austrian Institute of Technology for numerous internal discussions imparting the idea of in-depth investigation in the field of threat modelling and threat intelligence. In fact, AIT's input rekindled my long-standing interest in the topic and led to pursuing and extensive research project whose results have been laid down in the present thesis. Therefore, my gratitude extends to Korbinian Christl BSc. for intensive subject-related debates, to Dipl.-Ing. Dr. Mateis Cristinel for supplying inspiring elementary sources to get started, and to Christoph Schmittner MSc. for his tremendous support and highly appreciated reviews.

Special thanks go to my father Mag. Peter Chlup for investing countless hours into proof-reading my thesis and identifying typing errors with an eagle-eye and correcting linguistic ambiguities.

Table of Contents

List of Figures

List of Tables

1 Introduction

Current approaches of applying safety and security analysis already at design time have shown their effectiveness over the years. However, safety and security experts can hardly keep up with the amount of emerging threats that new components pose to the system under development (Ponemon, 2016). Hence, staying up to date with threats and possible countermeasures is becoming a vital part of the whole system lifecycle (Ma & Schmittner, 2016). For this reason, various cyber threat intelligence platforms have been developed. They range from simple blacklists of IP addresses (Li et al., 2019), over vulnerability databases, towards multi-step attack records (Jaeger, Ussath, Cheng, & Meinel, 2015). These databases contain up-to-date information on threats and vulnerabilities posed to the system.

We live in a highly interconnected society where human computer interaction plays an important role. Intrusion may yield safety, financial, operational and performance impact. Our systems are getting steadily more complex and, consequently, harder to maintain. The more technical components they contain, the bigger the attack surface (Schmittner et al., 2019; Strobl et al., 2018). As a result, our devices require a high amount of agile security. The static approach of security measures and analysis during design time is outdated and cannot cope with the ever evolving new generation of threats as they are becoming more manifold and complex.

The ongoing digitalization has its effect on arbitrary sectors. A prominent example is the automotive sector where *"hackers seek to access electronic systems and data, threatening vehicle safety and consumer privacy"* (UNECE, 2020). Thus, there is a new UN regulation stipulating cyber-security for all new vehicle types which will become mandatory in 2022.

In order to cope with the fact that technological systems are becoming increasingly more complex and new threats pop up every day, threat modelling has been developed. It enables the identification of security risks within a system model, more specifically a data flow diagram representing the system under consideration. This system model is checked against an abstract model which contains potential threats for various components and their configurations. The result is a catalogue containing threats and vulnerabilities that facilitate risk treatment. (Schmittner et al., 2019)

Threat modelling can be used with a digital twin of the actual system. The current approach for threat analysis is to adapt the abstract threat model by hand which takes a huge amount of time and effort before an analysis can be conducted. In addition, threat modelling is mainly utilized during the concept phase and is hard to adapt to new threats. Consequently, there are numerous improvements to be made.

In contrast to a digital twin of a system, which is highly agile and can be changed dynamically, the threat model which is a digital twin of potential threats and vulnerabilities is relatively static (Shostack, 2014). The proposed approach is automating this part by connecting to existing threat intelligence, more precisely vulnerability databases that can be found on the web and extract information to extend the abstract threat model. This way, new vulnerabilities can be integrated into the digital twin, both during development and maintenance phase. Whenever a system component changes or new threats are reported, new vulnerability information can be detected by rerunning the analysis. This not only enables safety and security experts to monitor the system during the whole system lifecycle, but also gives them a chance to review the system architecture with regard to the detected vulnerabilities.

On the one hand, this kind of automation boosts the effectiveness of threat analysis. On the other hand, the approach makes it easier to stay up-to-date with threats (Syed, Padia, Finin, Mathews, & Joshi, 2016). Finally, automatic information extraction will play an important role in reducing the number of cyber-security vulnerabilities when developing and maintaining a system.

1.1 Goals

The main objective of this master thesis is the reduction of cyber-security vulnerabilities within systems. It shall be possible to not only apply security measures during the design phase but during all phases of the product lifecycle. Threat modelling shall become capable of taking into account real-world vulnerabilities on top of common generic threats. This shall be done by integrating data from arbitrary sources in order to achieve greater coverage of known vulnerabilities with the ultimate goal of making systems more resilient. Thus, this thesis will add value to an existing threat model by including up-to-date information on vulnerabilities which will be considered during each phase of the product lifecycle.

1.2 Envisaged Solution

Based on the previously specified goals, this master thesis will provide an approach on how to automate the enhancement of existing threat models. It will integrate information from arbitrary sources such as NVD and Packetstorm. While some data is provided in a structured format, some information can only be obtained as plain text. This is why the approach proposed here will rely on extracting structured data as well as a Stanford Named Entity Recognizer trained on NVD and Packetstorm entries. This leverages the capabilities of threat modelling by including real-world information on vulnerabilities. Moreover, this master thesis will discuss a model for representing information about technological products and their associated vulnerabilities. Furthermore, it will show a possible mapping from CVSS to an ISO/SAE DIS 21434 based risk matrix and propose a possible solution to automatically create content for threat models.

1.3 Structure of the Thesis

The following section, 2, will concentrate on the scientific research work related to this thesis which served as inspiration. In addition, the topic of threat intelligence, as well as its challenges and reasons for (not) sharing information will be discussed. Furthermore, different exchange formats and platforms have been analyzed. As information needs to be extracted from plain text, various approaches regarding information extraction will be elaborated on.

Section 3 focuses on the topic of threat modelling and elaborates on the technology that AIT's threat modelling approach already builds upon. Moreover, this section discusses joint approaches for gathering vulnerability information, and illuminates enabling technologies.

This is followed by a discussion on a model created in order to represent products with their respective vulnerabilities and methods to convert existing metrics into a fitting representation. These facts and findings will be covered in section 4.

Afterwards, in section 5 the implementation of the prototype will be explained including how a connection to existing vulnerability databases is established, how the data is extracted and the way in which a named entity recognizer has been trained. This section will be concluded with an evaluation of the results.

Section 6 will elaborate on the experience gained during the writing and the implementation processes. Finally, section 7 will provide a recapitulation of the discussed topics in a conclusion.

2 Related Work

As the topic of this master thesis is the automatic extraction of information from vulnerability databases, literature research was mostly conducted in three main areas. First of all, the topic of threat intelligence including the different types of threat intelligence were of major interest as they form a level above vulnerability databases and vulnerability intelligence. Definitions of threat intelligence, reasons for sharing threat/vulnerability information and resulting challenges were included in the research.

The second main area included research on existing threat information exchange formats and platforms that integrate them. It is about differences of threat exchange formats and the kind of information they provide. This master thesis focuses on information that is valid for longer time periods. Consequently, more emphasis was put on platforms that provide threats and vulnerabilities which pose long time threats, in contrast to fluctuating threat information such as IP blacklists and file hashes. Moreover, the way in which threat information is presented was an important aspect when deciding on which formats and as a result, exchange platforms to focus on. While some formats rely on structured information, which can be integrated automatically, other formats include unstructured data, which requires further analysis and cannot be utilized directly.

The aspect of unstructured data leads to the third and last main research area: information extraction from unstructured text. More precisely, the topic of Natural Language Processing (NLP) (Jurafsky & Martin, 2000; Bird, Klein, & Loper, 2009) was included in the research. With these main areas covered, also papers yielding combinations of the prior were consulted.

The following sections will elaborate on these research areas in more detail.

2.1 Threat Intelligence

According to Tounsi and Rais, cyber threat intelligence is evidence-based knowledge about threats which can contribute to decision-making with the *"ultimate goal of preventing attacks or at least shortening the window between compromise and detection"* (Tounsi & Rais, 2017, p. 214). Chismon and Ruks define threat intelligence as follows: *"Information about threats that could inform decisions is arguably threat intelligence"* (Chismon & Ruks, 2015, p. 6). Li et al. describe threat intelligence as *"contemporary threat data sources"* (Li et al., 2019, p. 851). The Kaspersky company gives a stricter definition of threat intelligence. They draw a line between threat data and threat intelligence. In contrast to threat data which may just be a simple list of possible threats they consider it threat intelligence when *"IT specialists or sophisticated tools read threats and analyze them."* (Kaspersky, 2020) This analysis includes historical knowledge and possible countermeasures for a specific problem.

This master thesis defines the term of threat intelligence similar to Chismon and Ruks. More specifically in this thesis, any information that could improve system security in terms of common vulnerabilities, threat data or possible countermeasures is considered threat intelligence.

According to ENISA threat intelligence is critical for security in organizations and originates from incident response (ENISA, 2017). It should be integrated into an organization's workflow.

A yearly study of the Ponemon Institute (Ponemon, 2019) mentions that 74% of 1,098 IT security practitioners include threat intelligence in their organizations cyber-security practices. The usage of threat intelligence not only supports decision making and improves the resilience of a system, it also illuminates the risk landscape (Chismon & Ruks, 2015) for safety and security experts. Threat intelligence can be derived from various sources: from technical sources such as vulnerability scanning and analysis on the one hand, from organizational or human sources on the other. (ISO/TC 22/SC 32, 2020).

Threat intelligence is evolving rapidly. The number of newly detected threats rises drastically every year. Particularly, in large organizations the interest for threat information is on the increase (ENISA, 2018). Cyber attacks require new kinds of security defences. Attackers tend to exploit knowledge about known vulnerabilities or possibly even configuration errors in hardware or software while innovating new means of attacks. On the other side, the defenders improve their defensive measures and thus their resistance to certain attacks. This is done by utilizing information about possible threats in order to keep their security practices up to date and forge defensive strategies. (Tounsi & Rais, 2017)

2.1.1 Relevance of Threat Intelligence

Systems are constructed of diverse components with all sorts of communication channels. But more means of communication also result in a bigger attack surface (Schmittner et al., 2019; Strobl et al., 2018). To counter this, gathering of threat information is required to ensure security.

Although many organizations collect threat intelligence, the amount they can gather on their own is limited. Up-to-date information about threats is required to prevent attacks or at least detect them promptly in order to achieve timely disaster recovery (Li et al., 2019). New threats appear on a daily basis and it is almost impossible to process and utilize this information alone (Wagner, Dulaunoy, Wagener, & Iklody, 2016). Malware can bypass firewalls and anti-virus systems as these rely on signature based pattern matching. Simply changing one bit in the malware can lead to a different signature and possibly pose a new threat. Another important aspect are zero-day vulnerabilities which are hard to spot and often not found until months or even years pass (Tounsi & Rais, 2017). Moreover, threat actors are getting more organized. Consequently, cyber-security experts try to lower the security risks for their organization (Wang &

Chow, 2019). By utilizing the newest data on threats and vulnerabilities, and feeding them into the security applications they can speed up their response.

Sharing is a trend in the IT community. Some companies such as IBM, Crowdstrike, Kaspersky and CISCO provide threat intelligence commercially (Li et al., 2019). While e.g. anti-virus and intrusion detection companies sell threat intelligence as a by-product, others focus on threat intelligence exclusively.

However, threat intelligence can also be acquired for free. Open Source Intelligence (OSINT) collects publicly available information on threats (Vadapalli, Hsieh, & Nauer, 2018). Among them are Common Vulnerabilities and Exposures (MITRE, 2020a), National Vulnerability Database (NIST, 2020b), AlienVault OTX (*AlienVault*, 2020) and MISP (*MISP*, 2020).

Although new threats need to be identified fast, they also require accurate classification. Simply posting a threat without further analysis is generally not enough and may lead to false positives, which is also a major challenge for threat hunting teams (Ponemon, 2019). Reliable information sources are required.

2.1.2 Types of Threat Intelligence

Chismon and Ruks define 4 types of threat intelligence (Chismon & Ruks, 2015):

1. Firstly, **strategic threat intelligence** yields high level information and covers financial impact of cyber attacks. It can be used to understand current risks and identify new risks. Strategic threat intelligence is usually passed on via reports and conversations. According to Chismon and Ruks, strategic threat intelligence has an impact on the decision whether an organization should enter a certain marketplace, when considering a potential government hacking competitors.

2. Secondly, **operational threat intelligence** deals with details of specific attacks. This information is very rare as it covers knowledge of who might be going to attack an organization, when they might attack, and how the attack might happen. For this reason, only governments have access to the infrastructure to collect this type of information.

3. The third type of threat intelligence is **tactical threat intelligence**. It covers attacker methodologies, the tools they use, and their tactics. Moreover, it serves as a means for preparing an organization's infrastructure against current tactics and to reduce vulnerabilities. This information is consumed by defenders and incident responders.

4. Finally, **technical threat intelligence** is about low level information. Usually automatically consumed by machines, it plays an important role in monitoring of malicious activity within an organization. An example for this kind of information is a list of IP addresses considered as malicious.

While strategic and tactical threat intelligence are suited for long term use, operational and technical threat intelligence represent information that is valid only for short periods of time. But in order for companies to brace themselves against potential attacks this information is not less important. Strategic and operational threat intelligence are kinds of high level information that need to be considered when planning a technological system. By contrast, tactical and technical information represent the low level side of the medal. With this low level information, organizations can implement specific countermeasures into their systems (Chismon & Ruks, 2015).

Miller distinguishes only 3 types of threat intelligence. He defines strategic threat intelligence as the "who" and "why", operational threat intelligence as the "how" and "where" and tactical threat intelligence as the "what" (Miller, 2018).

Nevertheless, all this information needs to be integrated into an organization's security workflow. Due to the highly complex nature, the enormous amount and the fluctuating durability of information many challenges arise. The next section gives an insight into these challenges.

2.1.3 Challenges of Threat Intelligence

Vadapalli et al. point out that the manual methods of open source intelligence collection and analysis are inadequate because of the volume of newly arising threats (Vadapalli et al., 2018). On the one hand, information on new threats and vulnerabilities must be integrated as fast as possible in order to prepare timely responses. On the other hand, this information may become invalid after a certain time period (hours, days, months) and is often not useful for attack prevention afterwards (Tounsi & Rais, 2017). The duration of usefulness of threat information fluctuates strongly. When considering IP address blacklists the information may only be valid for a very short time period, especially when it comes to malware that is utilizing cloud computing services. Also file hashes which are checked by anti-virus software change at a very fast pace. As a consequence, the systems need to adapt themselves all the time.

When it comes to zero-day exploits in software, the validity of the threat information increases. Customers utilizing a version of the software that is affected by the exploit can download a patch to improve the system resilience. But also hardware vulnerabilities represent long term valid vulnerability information.

As far as threat intelligence is concerned, companies - and as such security experts - need to distinguish what is important for their needs (ENISA, 2014a). To be more precise, companies need to decide if certain threat information is relevant to the application domain (Sillaber, Sauerwein, Mussmann, & Breu, 2016). *"Indicators of compromise will generally be considered relevant when a threat could affect the recipient's system"* (ENISA, 2014a, p. 3). Analysis of the data takes time, and even though incidents are discovered, they may be published only weeks or even months later. This is usually done to ensure accuracy and completeness of the relevant information. The information should be immediately consumable after undergoing verification, being as complete as possible. Apart from this, legal issues may affect the completeness of the data due to privacy issues (ENISA, 2014a).

When talking about threat information exchange, the topic of trust is inevitable. Trust towards the information sharing platform of choice should be guaranteed. This has to be put into practice not only in terms of data quality, meaning that the information should be *"reliable and credible"* (ENISA, 2017), but more importantly in access restrictions. In addition, platforms need to ensure that only authorized participants may access the shared data. This is to ensure that potential attackers do not get in touch with the published information and consequently exploit potential vulnerabilities (Sauerwein, Sillaber, Mussmann, & Breu, 2017).

Li et al. introduce threat intelligence metrics. They measure the volume (the amount of information different threat intelligence providers offer), the differential contribution (the delta in the data different providers share), the exclusive contribution, the latency (the time period between the first platform publishing and other platforms also updating their content), the coverage and the accuracy of entries. These are measures that may be used by companies in order to decide which platform to retrieve their threat information from. Li et al. take measurements and present the measurement results alongside an analysis. Their paper deals mostly with IP blacklists and file hashes which belong to the most common forms of threat intelligence utilized in firewalls and anti-virus software but has no real impact on the topic of threat modelling at the component level and the whole system lifecycle.

2.1.4 Sources of Threat Intelligence

ISO/SAE DIS 21434 contains possible sources of threat intelligence. These sources can be divided into two categories. On the one hand, threat information can come from external sources such as researchers, government sources or customers, as well as from commercial and non-commercial sources such as threat and vulnerability databases. On the other hand, internal sources about configuration information of a system, field information through vulnerability scanning or repairs and results of vulnerability analysis can give security designers an insight into the threat landscape. (ISO/TC 22/SC 32, 2020)

2.1.5 Reasons for Sharing

In order to keep systems safe and secure, information sharing plays an essential role (NIS Directive, 2016). But it comes with a lot of challenges. Tounsi and Rais point out reasons for information sharing. First of all, information about possible threats and attacks that lie in the past can be used to prevent or handle potential cyber attacks in the future. Tounsi and Rais claim that sharing threat intelligence accelerates threat detection. Furthermore, the shared information yields insight into the way threat actors operate (Tounsi & Rais, 2017). This way security experts can raise the situational risk awareness of their organization, decreasing the *"likelihood of cascading effects"* (Zheng & Lewis, 2015, p. 1) in a system. In order to protect systems thoroughly, organizations need to include threat intelligence into their incident-, vulnerability-, and risk-management strategies (ENISA, 2018). The platforms for information sharing are sufficient when low confidentiality incidents are reported, as in the Cyber Vault Project. (ENISA, 2018; *The Cyber Vault Project*, 2020)

2.1.6 Reasons for Not Sharing

Sillaber et al. deal with the difficulties of information sharing (Sillaber et al., 2016). ENISA; Tounsi and Rais and Chismon and Ruks point out why some companies do not share their information (ENISA, 2018; Tounsi & Rais, 2017; Chismon & Ruks, 2015). One of the major points for not sharing threat or vulnerability information is negative publicity. A company posting information about a vulnerability in their system publicly, may negatively influence their market share (Tounsi & Rais, 2017; Chismon & Ruks, 2015). NIS Directive encourages information sharing and points out that *"it is essential to ensure that operators of essential services and digital service providers who participate in such exchanges are not disadvantaged as a result of their cooperation"* (NIS Directive, 2016, § 35).

Another important aspect are legal rules and privacy. The unawareness of laws and regulations in different states stops entities from sharing. Many people do not know what information they are allowed to share. Apart from this, some countries consider threat information sharing a crime (ENISA, 2018). Quality issues are also a common reason for not sharing. The information may not be specific enough or too old to be considered actionable. Also the data format plays an important role. If an organization is unaware of an attack they will not publish information, as no security incidents were detected. (Chismon & Ruks, 2015; Tounsi & Rais, 2017)

Finally, not sharing may also be the result of authorities holding back information about product vulnerabilities as there is no security fix yet available (NIS Directive, 2016). Potential attackers could exploit this information knowing that a certain vulnerability has not been acted on.

2.2 Threat Information Exchange

Threat information is posted in different formats and structures (Wang & Chow, 2019). It is shared via various platforms such as social media, news reports and dedicated platforms for threat intelligence, some commercial and some open source (Wang & Chow, 2019; Li et al., 2019). Some of them disseminate threat data in unstructured plain text (Ramnani, Shivaram, Sengupta, & M., 2017; Sillaber et al., 2016). Others utilize standardized representations of threat intelligence information. The latter yield better quality as information is structured and can be processed more easily by computers in an automated manner. It is not only important to share threat data fast to avoid attacks, but also to ensure the feasibility of utilizing this information without losing quality.

2.2.1 Formats

Different standards and formats for threat exchange have been developed over the years (ENISA, 2014b; Menges & Pernul, 2018; Tounsi & Rais, 2017; Sauerwein et al., 2017). Ranging from enumerations towards incident response formats for threat exchange, different platforms rely on different standards. ENISA and Menges and Pernul discuss these different standards in detail (ENISA, 2014b; Menges & Pernul, 2018). An overview of the ones considered the most important will be discussed shortly in this section.

Common **A**ttack **P**attern **E**numeration and **C**lassification (CAPEC) has been developed by MITRE (MITRE, 2020e) and represents a catalogue describing the techniques and procedures utilized during a possible attack. It is used in STIX and IODEF which will be explained later.

Also developed by MITRE are CWE and CVE. Actually, CWE, the **C**ommon **W**eakness **E**numeration is a *"list of commonly occurring software weaknesses and vulnerabilities"* (ENISA, 2014b, p. 12). It aims at avoiding vulnerabilities when building a system. CWE is used in IODEF and STIX.

Probably, the most commonly known enumeration by MITRE is CVE - **C**ommon **V**ulnerabilities and **E**xposures. It yields unique identifiers that reference vulnerabilities and includes a description about vulnerabilities and exposures of various components. CVE is utilized in a large number of cyber-security products and is the industry standard for vulnerability identification (MITRE, 2020a). It represents a standardized basis for the evaluation of vulnerabilities and exposures, and offers a description for each of them.

As pure CVE is typically not enough for direct processing and is not directly ingestible without further transformation, NIST (NIST, 2020a) developed the **N**ational **V**ulnerability **D**atabase (NVD). NVD builds on top of CVE. Consequently, every CVE entry is included in NVD. Moreover, the NVD yields additional information for CVE entries. It extends a CVE entry with CPE for product identification, CWE for identification of the weakness, it assigns a CVSS (explained below) and provides hyperlinks to advisories and fixes (MITRE, 2019a). STIX, IODEF and VERIS, among many others reference CVE entries.

Common **P**latform **E**numeration (CPE), which has already been mentioned before, has been developed by NIST, and represents a naming scheme for products in order to identify them appropriately, especially in terms of threats and vulnerabilities. IODEF and NVD rely on this information.

FIRST (FIRST, 2020a) developed a **C**ommon **V**ulnerability **S**coring **S**ystem (CVSS) capable of rating vulnerabilities with scores ranging from 0 - least critical, to 10 - most critical, which is used by many of the above formats.

Following enumeration formats, standardized incident reporting formats will be discussed. Incident reporting formats have been developed to share information on threats in various ways. The most common is probably STIX (*OASIS CTI TC*, 2020), the **S**tructured **T**hreat **I**nformation e**X**pression. It is used by over 40% of the organizations that include threat intelligence in their security processes (Menges & Pernul, 2018) (Ponemon, 2019). STIX, in its current version 2.1 has been developed by OASIS CTI (*OASIS CTI TC*, 2020). It represents a standardized opportunity to share information concerning security issues in a holistic way using the JSON format as representation. STIX contains **I**ndicators **o**f **C**ompromise (IoCs). IoCs indicate a possible intrusion into a system and identify potentially malicious activities. They may be malicious files, IP addresses that are considered malicious or atypical user activities (Sauerwein et al., 2017). The purpose of STIX not only resides within threat information sharing, but also also in prevention and response strategies against certain threats (ENISA, 2014b). Due to its complexity, STIX requires software in order to support efficient human readability, but therefore is easily ingestible for machines.

IODEF, the **I**ncident **O**bject **D**escription **E**xchange **F**ormat, currently in version 2.0, is a format that includes attack patterns, vulnerabilities, weaknesses, platforms and scores. To be more precise, it links to CAPEC, CVE, CWE, CPE and CVSS. IODEF is a semi structured format. It uses XML tags which in some cases include plain text (Menges & Pernul, 2018). Consequently, it requires manual analysis by domain experts. IODEF has been developed by IETF (IETF, 2020).

Another commonly used form of threat exchange is the **V**ocabulary for **E**vent **R**ecording and **I**ncident **S**haring (VERIS) framework (*The VERIS Framework*, 2020) which is used for sharing incident reports and also includes metrics. VERIS contains a structured language for security incidents.

A further kind of information sharing is the **A**buse **R**eporting **F**ormat (ARF) and its extensions MARF and X-ARF (ENISA, 2014b). They extend MIME and are intended to share spam or incident reports via email. Their main strength is their inherent low complexity, which results in good human readability. Therefore, reports require manual analysis. (Menges & Pernul, 2018)

2.2.2 Platforms for Threat Exchange

Sauerwein et al. analyze 22 threat intelligence sharing platforms and claim that STIX is the de-facto standard for describing threat intelligence (Sauerwein et al., 2017). Moreover, the authors point out that most of the platforms that offer threat intelligence only collect the data. The responsibility for the analysis and the integration of the data still resides with the consumers. The platforms mostly focus on the IoCs, which is information giving insight into possible intrusion and malicious activities (Sauerwein et al., 2017).

One platform that is not listed as threat intelligence platform in most of the literature, but yields important information for incident exchange, as it is often included within formats, is CVE. Its description includes information on vulnerabilities of system components like hardware or software. With NVD which builds on top of CVE, the data can become quite informative in terms of common weakness enumeration and possible fixes (MITRE, 2019a).

(ENISA, 2017) provides a collection of current threat intelligence platforms distinguishing between the type of solution - open source, commercial or community. **C**ollaborative **R**esearch **I**nto **T**hreats (CRITs), the **M**alware **I**nformation **S**haring **P**latform (MISP) and the **C**ollective **I**ntelligence **F**ramework (CIF) focus on the domain of open source threat sharing, to name a few.

When it comes to commercial solutions for threat intelligence, scoutTHREAT (LookingGlass, 2020), ThreatConnect (ThreatConnect, 2020) and ThreatStream (Anomali, 2020) can be named. In contrast, AlienVault Open Threat Exchange (OTX) (*AlienVault*, 2020), Facebook Threat Exchange (Facebook, 2020) and X-Force Exchange (IBM, 2020) form community solutions. A listing of threat information exchange platforms can be found in (ENISA, 2017).

2.3 Information Extraction

In terms of threat intelligence representation, some formats support structured information which is machine readable. Other formats such as articles or descriptions in plain text work with unstructured data. Although unstructured data is understandable to humans, the uptake of threat information is slow due to lacking automation mechanisms. Machines cannot utilize the information to perform specific tasks directly. Unstructured text requires conversion to a more compact representation that computers can work with. Consequently, new mechanisms need to be developed for the extraction of cyber-security information from unstructured text. This will speed up the pace of security measures for a system (Syed et al., 2016). Natural Language Processing (NLP) offers sophisticated information extraction techniques that can be used for this task (Jurafsky & Martin, 2000; Bird et al., 2009).

According to Wang and Chow, only a limited number of keywords in a sentence is relevant. All irrelevant information should be omitted while extracting only the significant information. Wang and Chow mention that extraction from unstructured data forms a challenge, as there is no fixed pattern and every human has a different writing style. In (Wang & Chow, 2019) the authors use the

Java Annotation Pattern Engine (JAPE) and the General Architecture of Text Engineering (GATE) framework to extract information from threat intelligence articles and reports which were then stored inside an ontology.

Another approach in information extraction from unstructured information was conducted by Vadapalli et al.. The authors utilized the Twitter Streaming API to download tweets with relevance to cyber-security. The Stanford CoreNLP which is widely used in the Natural Language Processing (NLP) domain was used to perform the extraction tasks. In the first step, experiments were conducted without having a dedicated model trained for cyber-security and resulted in poor performance. In the next step, the stucco entity-extractor library (*stucco/entity-extractor*, 2020) was included in the NLP tasks. This library was implemented to retrieve cyber-domain entities from unstructured text. Vadapalli et al. also describe the libraries used for the experiments, which enables the reproducibility of the results.

Mulwad, Li, Joshi, Finin, and Viswanathan developed a prototype which is able to transform unstructured text into RDF/OWL. They use an SVM (Support Vector Machine) classifier to filter out descriptions which contain information about vulnerabilities. With Named Entity Recognition (NER), relevant entities were extracted. Afterwards the resulting entities were mapped in OWL (Mulwad et al., 2011). Joshi, Lal, Finin, and Joshi analyze key concepts from the unstructured description field of CVE entries using a conditional random fields (CRF) algorithm (Finkel, Grenager, & Manning, 2005) from Stanford NER , which is pre-trained to identify people, organzations and places. Additional training was conducted with data from security blogs, CVE descriptions and security bulletins. For data representation Joshi et al. use the Intrusion Detection System (IDS) ontology (Pinkston, Undercoffer, Joshi, & Finin, 2003) and map the retrieved information to DBPedia (*DBpedia*, 2019) entries.

Satyapanich, Finin, and Ferraro utilized Stanford CoreNLP for word embedding. They conducted experiments with two different approaches, this is to say one with word2vec (Mikolov, Chen, Corrado, & Dean, 2013) and another one with BERT (Horev, 2018). The Unified Cybersecurity Ontology (UCO) (Syed et al., 2016) was used to represent the extracted data in a knowledge graph. (Satyapanich et al., 2019)

Mittal, Joshi, and Finin introduce Cyber-All-Intel which consists of a data collection engine that collects information from multiple data sources. After preprocessing, a Security Vulnerability Concept Extractor (SVCE) is applied to the collected resources. Afterwards the SVCE retrieves cyber domain entities and their relations and stores them in a knowledge graph based on UCO. They build two applications - firstly, a mechanism to query the knowledge graph, secondly, an application to generate alerts based on system profiles. (Mittal et al., 2019)

The approach in (Pingle, Piplai, Mittal, & Joshi, 2019) for extracting cyber-security entities from unstructured text also depends on an NER system, which was created in (Mittal, Das, Mulwad, Joshi, & Finin, 2016). After conducting the entity extraction step, word2vec (Mikolov et al., 2013), which was trained on a cyber-security corpus, served for generating vector embeddings of the retrieved data. Moreover, Pingle et al. discuss the representation of information in knowledge graphs and make use of UCO.

Ramnani et al. build an information extraction system and use the STIX format as a base for information representation. They follow 3 steps: firstly, the integration of cyber-security sources, secondly, the categorization of the retrieved information, and thirdly, the analysis of trends and time-series by filtering out the information that is relevant for an organization.

Ramnani et al. propose a semi-automated information extraction technique. Basilisk (Thelen & Riloff, 2002) was applied for pattern detection and NER for the entity extraction task. The resulting pattern detection system was used for extracting information from new documents (Ramnani et al., 2017).

Syed et al. extract information from NVD, generate RDF triples therefrom and map the information to DBPedia entries. The authors utilize stucco extractors for entity extraction (Syed et al., 2016)(*stucco-archive/extractors*, 2019).

2.4 Starting Points for the Master Thesis Identified in Literature Research

In modern connected society security represents one of the major challenges. Threats and their corresponding security measures affect all kinds of organizations in the same way. Every company is exposed to some extent. Consequently, threat intelligence and vulnerability information sharing between different entities represent a necessary form of exchange in order to make technological systems more resilient. We have defined threat intelligence as any information that could improve our systems' security, listed reasons for information sharing and analyzed why some companies may not be willing to share their data. Also trust towards the utilized platform, the participants and the information presented plays an important role for potential consumers of threat intelligence, which is why different sharing levels exist. Moreover, the different types of threat intelligence and their consumer groups have been identified. A variety of formats for threat information exchange have been presented, of which the most commonly used is STIX. When it comes to vulnerabilities, the NVD provides a powerful resource.

However, most of this information is still shared in unstructured plain text, such as reports or blog posts. Consequently, the shared information is not directly ingestible by machines. In order to process this information NLP can play an essential role. Most of the publications that were examined utilized a Named Entity Recognition approach to categorize the retrieved information and map it to a format suitable for representation such as the Unified Cybersecurity Ontology (UCO)(Syed et al., 2016).

Threat intelligence can serve as a powerful tool to improve threat modelling. Additional information on vulnerabilities and intrusion systems facilitates the development of an abstract model containing the threat information. Analysis can be conducted in more detail without extensive manual configuration. An automated approach to integrate threat information can make a vital contribution in reducing cyber-security vulnerabilities when developing and maintaining a system.

3 Basic Technological Framework

The topic of cyber-security is becoming a rising challenge in modern application development. Hardware and software are confronted with an ever changing risk landscape and developers as well as safety and security analysts need to be aware of threats that the system under consideration is exposed to. Hence, security must be considered during the whole design process in the same way as application developers consider usability and performance (Torr, 2005). We need to design our hardware and software in a way that they can withstand attacks in the future. In order to achieve more resilience, threat modelling which is closely related to risk management approaches can be applied.

The following sections will elaborate on the concepts behind threat modelling based on an example. Moreover, the threat modelling components, namely, system model, threat model and the threat analysis will be focused on. Afterwards, the closely intertwined topic risk management including risk assessment and risk treatment will be discussed and brought into relation with threat modelling. Finally, the concept of rules which constitute the threat model will be explained.

3.1 Threat Modelling

Threat modelling has become an important tool in identifying potential safety and security risks. Moreover, it helps in detecting weaknesses of the architecture of the system and can reveal possible design flaws. According to Torr *"Threat modelling should be treated like any other part of the design and specification process"* (Torr, 2005, p. 66). Applying countermeasures in the early stages of product development reduces costs.

This master thesis will go a step further and consider threat modelling not only during the design phase but also during the operational and maintenance phases. It proposes that threat modelling should be stuck to during the whole system lifecycle. This way, all the knowledge gained during later phases can be integrated into the system in order to mitigate certain security flaws and to make the system more resilient in the future.

As threat modelling serves the purpose of identifying potential safety and security risks, it can also help with finding mitigation strategies and countermeasures towards certain identified threats. Laorden, Sanz, Alvarez, and Bringas claim that threat modelling provides *"useful guidelines on how to mitigate the associated risks"*(Laorden et al., 2010, p. 2). This is something that the improved threat modelling approach will build upon, by providing links to references and solutions addressing certain vulnerabilities of hardware or software components, as well as pre- and postconditions. The NVD and Packetstorm serve as the main sources of information.

The following subsections will provide an overview of the threat modelling process, and discuss current shortcomings and possible solutions.

3.2 Threat Modelling Process

Threat modelling is an iterative process. Its central constituents are the system model and the threat model which are compared to each other. This results in a catalogue of threats relevant for risk treatment. But this is not where it ends, threat modelling allows the application of countermeasures to a system representation based on the identified risks. Consequently, a new comparison of system and threat model must be conducted. The process is illustrated in figure 1.

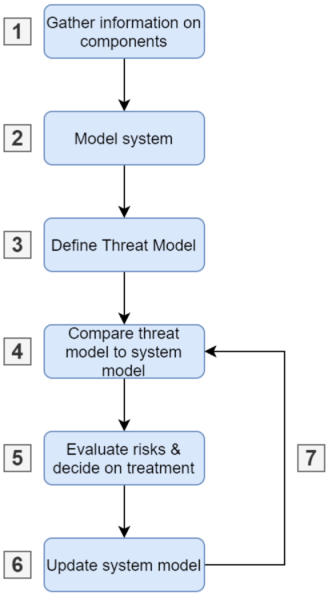

1
Gather information on components
2
Model system
3
Define Threat Model
4
Compare threat model to system model
5
Evaluate risks & decide on treatment
6
Update system model
7

Fig. 1. An illustration of the threat modelling process based on (Schmittner et al., 2019)

An additional listing of the steps constituting the threat modelling process is given below based on (Schmittner et al., 2019):

1. Gather information on components to be used or in case of an already running system, existing components
2. Model the system with all security assumptions

3. Generate threat model by isolating threats for components and derive them from available sources
4. Compare the threat model to the system model in order to identify potential threats
5. Conduct risk evaluation on all identified threats and decide on the risk treatment
6. Adapt the system model with mitigation strategies and security countermeasures
7. Go back to step 4 to identify newly introduced threats or threats that were left untreated

Now that the general steps of threat modelling have been pointed out, the following sections will provide more detail.

3.2.1 System Model

The first step in threat modelling requires gathering information on components. Its objective is to check which components should be used when designing the system, or for an already running system which components are already integrated. Also the type of communication between components needs to be analyzed and included inside the model. All this information gathering should be done as exact and with as much detail as possible (Torr, 2005). All the safety and security assumptions that are known should be included.

The next step in threat modelling is to find a suiting representation which serves as a basis for identifying possible weaknesses. In this context the information mentioned above resembles a system model.

The approach by AIT uses a data flow diagram for representing the system model. A data flow diagram is capable of modelling a technological system graphically. This includes a definition of components, their connections and configuration. This modelled information can be used to conduct an analysis for a certain component or a configuration of components (Torr, 2005; Ma & Schmittner, 2016). The process of transforming the concept or real-world information of an application into a graphical representation can already influence the system design and reveal possible design flaws.

Consider a simple web-application as an example for the system model. It shall be capable of providing a web-frontend to the end-user that can be accessed over the internet. Moreover, it shall be capable of storing user generated content and allow the end-user to log in and out of the application.

Figure 2 shows a system hosting a simple web-application that is exposed to the end-user. The web-application is hosted by a web-server and all the communication is running over an internet connection. The web-server itself is connected to a database. They are located within a trusted environment boundary indicating that both, the web-server and the database trust each other. This system model will serve as a basis for later examples in this master thesis.

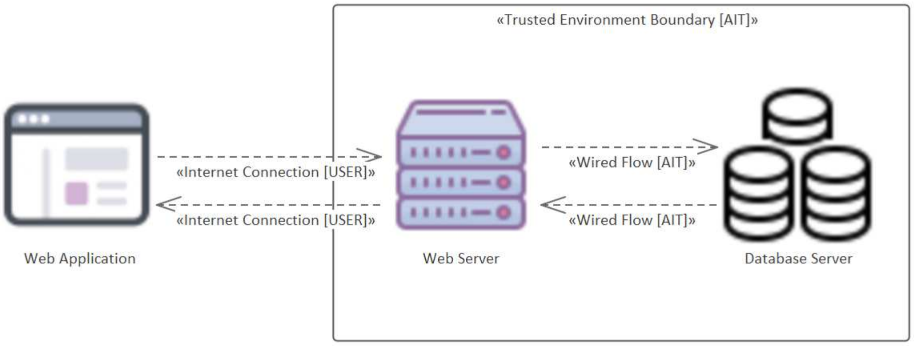

«Trusted Environment Boundary [AIT]»
«Internet Connection [USER]»
«Internet Connection [USER]»
«Wired Flow [AIT]»
«Wired Flow [AIT]»
Web Application
Web Server
Database Server

Fig. 2. An example of a simple illustrative system model

It is of utmost importance to make the representation of the technological system as exact as possible and not to create a representation of the system as it should be. The components, connections and their configurations need to be modelled with as much detail as possible in order to be able to detect real threats that may target the system. Simply designing the data flow diagram in a way the developer thinks that it should work is not sufficient and might leave existing threats unrecognized and untreated. (Torr, 2005)

3.2.2 Threat Model

After creating a system model and a representation that fits the requirements for the system, a second model needs to be established. In order to detect threats within the system model, this model must be capable of processing known weaknesses, vulnerabilities and threats. The resulting model is called the threat model. By applying the threat model to the system model potential threats can be identified. (Schmittner et al., 2019)

To achieve this, AIT utilizes a rule-based approach. Common configuration errors are modelled as anti-patterns. The term anti-pattern depicts a configuration that the system should not include. The rules resemble known configuration issues, e.g. in terms of communication through trust boundaries, missing encryption when using wireless communication or missing tamper protection. The information for these rules comes from various sources, such as UNECE WP29 (World Forum for Harmonization of Vehicle Regulations), ETSI (European Telecommunications Standards Institute) or the ITU (International Telecommunication

Union). The structure of such a rule is explained in section 3.4.

Also the example from before is exposed to potential threats. When considering the internet connection between the web-application and the web-server as shown in figure 3, attackers could manipulate their own IP address in order to disguise their actions as coming from a trusted source, also known as IP Spoofing. Moreover, the web-server itself might become target of a distributed denial-of-service (DDoS) attack without proper mitigation strategies.

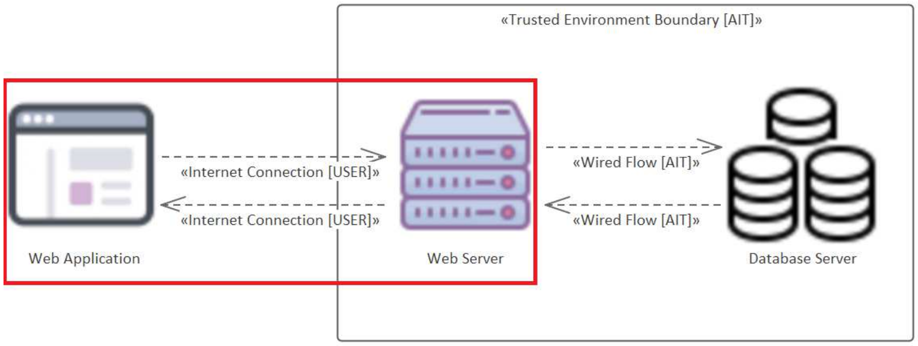

«Trusted Environment Boundary [AIT]»
«Internet Connection [USER]»
«Internet Connection [USER]»
«Wired Flow [AIT]»
«Wired Flow [AIT]»
Web Application
Web Server
Database Server

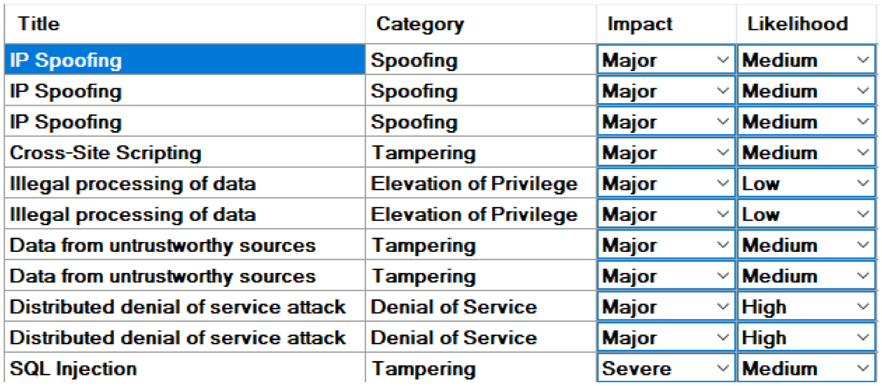
Fig. 3. Example of vulnerable components within the simple illustrative system model

All this information about possible threats is represented within the threat model. However, what is presented here are just examples of threats the system could be susceptible to. The results of a real analysis on the system are shown in the next section. They depend strongly on the configuration of the system model.

As the step of creating a representation of threats and weaknesses manually is time consuming and various systems utilize similar means of configuration, AIT decided to create a knowledge-base for this information for distinct domains. All the information deduced from the sources mentioned before is incorporated within one single knowledge-base which supports reusability and speeds up the analysis due to a shorter or even absent threat information gathering process. (Ma & Schmittner, 2016)

3.2.3 Threat Analysis

Once system model and threat model have been finalized, an analysis can be conducted. This analysis compares the system model with the threat model and discloses known threats. The result is a catalogue containing threats posed to specific components, communication channels or their compounds. Figure 4 shows the results of a threat analysis on the example system from the previous sections.



Title	Category	Impact	Likelihood
IP Spoofing	Spoofing	Major	Medium
IP Spoofing	Spoofing	Major	Medium
IP Spoofing	Spoofing	Major	Medium
Cross-Site Scripting	Tampering	Major	Medium
Illegal processing of data	Elevation of Privilege	Major	Low
Illegal processing of data	Elevation of Privilege	Major	Low
Data from untrustworthy sources	Tampering	Major	Medium
Data from untrustworthy sources	Tampering	Major	Medium
Distributed denial of service attack	Denial of Service	Major	High
Distributed denial of service attack	Denial of Service	Major	High
SQL Injection	Tampering	Severe	Medium

Fig. 4. Threat catalogue derived from an analysis of the system model

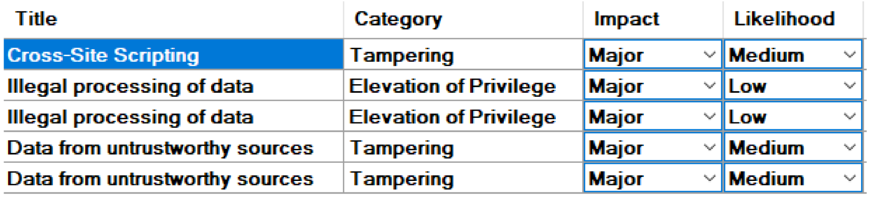
The resulting threat catalogue presents information on impact and likelihood, as well as the threat category. This information can be used to evaluate the identified threats and serves as basis for risk treatment. It helps to decide which threats and, consequently, risks should be treated and which should be omitted.

Please note that likelihood is actually a probability value and has too many parameters to be properly calculated. Therefore, this master thesis replaces it with exploitability which is easier to determine as it can be based on the technical description of the system under consideration.

In the current system no security properties have been set which means that there are no applied security measures. Consequently, there is a lot to be improved. When considering the IP Spoofing threats from before, authenticated connections could be used as a mitigation strategy. Moreover, a DDoS mitigation can prevent a distributed denial-of-service attack, while implementing input sanitization on the web-server before communicating with the database can eliminate a potential SQL injection.

After deciding on the risk treatment, system developers can apply countermeasures to their system and respectively their system model. Risk treatment has an effect on the existing system. Consequently, the next step after applying countermeasures and mitigation strategies requires a re-analysis of the system (Schmittner et al., 2019; Torr, 2005). Applying the security measures mentioned above reduces the threat catalogue as specified in figure 5.

The process can - as in the case demonstrated here, but does not necessarily - result in a more resilient system. This becomes clearer when considering the



Title	Category	Impact	Likelihood
Cross-Site Scripting	Tampering	Major	Medium
Illegal processing of data	Elevation of Privilege	Major	Low
Illegal processing of data	Elevation of Privilege	Major	Low
Data from untrustworthy sources	Tampering	Major	Medium
Data from untrustworthy sources	Tampering	Major	Medium

Fig. 5. Threat catalogue countermeasures applied

following examples:

A threat is detected for a certain component during the design phase. The developer knows that there is a specific component that already mitigates that threat and decides to integrate that component. What the developer might possibly not know is that it lacks other security features. Although the threat that was initially detected has been taken care of, other threats might appear due to missing security measures.

It may well happen that a hardware component within a running system with built-in tamper protection breaks and the respective component is not produced anymore. The system developers decide to substitute this component with a different one. However, the new component does not support tamper protection. Simply integrating this unit into the system without any analysis might introduce new threats. Although the general functionality of the newly introduced component may be the same, the two differ in their security features.

From this we can deduce that the threat modelling process is never finished. New threats emerge daily. Previously unknown vulnerabilities may be detected and exploited. Hence, staying up-to-date with the latest developments is the key to improved security. Analyzing the system with an underlying up-to-date threat model raises the awareness of new risks and provides means to monitor the system in the ever-evolving risk landscape.

The general concept behind threat modelling is closely related to risk management concepts. The following section will provide an excursion into the topic of risk management, risk assessment as well as risk treatment.

3.3 Risk Management

The aim of risk management is to identify, analyze, evaluate and, consequently, treat risks in a specific context. This approach is meant to reduce the risk for a system and the respective company behind it. (ISO/TC 22/SC 32, 2020)

Applying security to an application is always a trade-off between effort, consequences, exploitability and the related costs. This is why risks must be managed at all levels of an organization's structure including the will of stakeholders. It must be clear who is involved in the decision making process and how these decisions are made. (ISO/TC 262 Risk management, 2018).

Risk management is considered an iterative process due to the fact that the context of the application may change. New risks regarding threats appear daily, existing risks may change or even disappear. Hence, the system under consideration must be monitored constantly while identifying new risks that may affect it. Furthermore, a suitable risk management process provides aid in the identification of mitigation strategies as well as in informing decisions in order to allow timely responses. This is why it is also vital to integrate risk management with the system's objectives and to find ways on how to treat objectives that are in conflict with each other. (ISO/TC 262 Risk management, 2018)

The threat modelling approach that is covered in this master thesis utilizes risk management based on the ISO 31000 (ISO/TC 262 Risk management, 2018) and the ISO/SAE DIS 21434 (ISO/TC 22/SC 32, 2020). In the following, the approach proposed by these two standards will be compared with the approach at AIT. As risk management highly relies on individuals, each with their own perceptions and experiences, the results of the risk analysis and consequently, the risk treatment process arise from subjective points of view.

3.3.1 Risk Assessment

The risk assessment process can be split into three steps, this is to say risk identification, risk analysis and risk evaluation. (ISO/TC 262 Risk management, 2018; ENISA, 2020a)

Risk Identification describes the process of detecting risks that are interfering with the objectives of a company or application. It investigates possible causes of risk, vulnerabilities and threats. Additionally, this step considers the properties of assets. (ISO/TC 262 Risk management, 2018)
In our example from before, this step is closely related to the threat analysis step where the system model is checked for possible threats. When a rule fires due to the fact that a generic anti-pattern has been detected, this can be considered as a threat modelling based equivalent to risk identification.

Risk Analysis describes the step following risk identification. It deals with the exploitability of certain events and their impact on the system. Depending on the application domain, opinions and the viewpoint of an individual, the results of such an analysis may vary. (ISO/TC 262 Risk management, 2018)
As the threat model holds all relevant information concerning threats, the impact and exploitability levels, as well as the classification in STRIDE are predefined based on best practices. One could argue that defining risk levels beforehand is incorrect in terms of risk management. This objection is well justified, which is why the values of impact and exploitability reflected in the threat catalogue

from before and the risk matrix (which will be discussed in section 3.3.3), are configurable. The result should rather be considered a proposition than a fixed value. Different domains might require different approaches.

Risk Evaluation utilizes the outcome of the risk analysis as input. It is relevant when defining which risks need to be treated and how they shall be treated in terms of the applied methods (ISO/TC 262 Risk management, 2018). Concerning threat modelling, risk evaluation is the main factor when deciding whether a threat is relevant or, in other words, whether it is a potential threat.

As there may be multiple solutions for one and the same problem, a solution suitable for multiple weak-points may be considered. The goal of risk evaluation is to support the decision making process. While some risks need to be analyzed further, others may not require any action. Furthermore, the evaluation may reveal conditions that do not meet the original objectives and, therefore, lead to a re-evaluation of these objectives.

This final step of the risk assessment is currently not supported by the threat modelling approach in this thesis. It still requires interaction of individuals and relies completely on expert knowledge as well as research. However, this master thesis will add value by integrating online repositories and linking references to possible solutions and mitigation strategies. It will be capable of providing suggestions to experts in order to ease the risk evaluation process.

After conducting risk assessment action must be taken in order for the evaluation to take effect. This is done in a process called risk treatment.

3.3.2 Risk Treatment

Once possible strategies to reduce risk have been identified, experts need to decide on how certain risks are treated. This involves an evaluation of the effectiveness of certain treatment options. Moreover, experts discuss whether the residual risk is tolerable before implementing a treatment option. Should the risk be inacceptable, other treatment options might be considered or further treatment may be required (ISO/TC 262 Risk management, 2018). In terms of threat modelling, the effect of these changes is reflected by running an additional analysis.

There are four basic options when deciding on the risk treatment (ENISA, 2020b). Figure 6 holds an illustration of abstract risk treatment measures.

When a certain risk has a high exploitability as well as a high impact on the system, a company may decide to **avoid** that risk by adapting their system and eliminating the risk's cause. However, avoiding risk is only feasible if it does not interfere with the objectives of the organization.

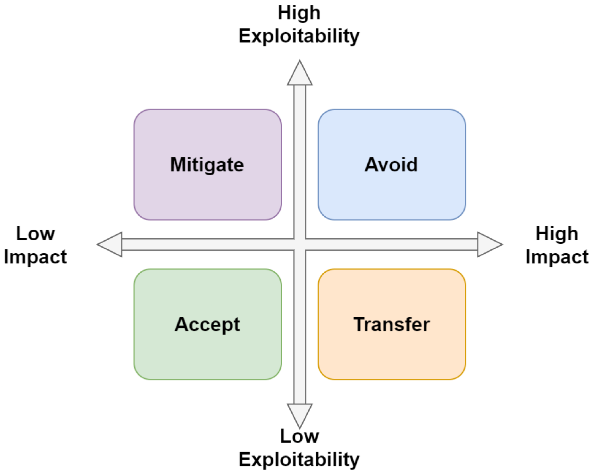


Fig. 6. Risk treatment with regard to exploitability and impact based on (theresilience, 2018), (ENISA, 2020b)

Risk can also be **transferred** to another organization. This might be considered when the exploitability is low but the impact is high. Transferring risk means e.g. getting an insurance or relying on well-known third party providers.

Should both impact and exploitability of an occurring event be low, companies might consider to **accept** the risk and its consequences.

Finally, risk can be **mitigated**, which is usually done when the impact is low but the exploitability is considered high. Modifying the system by applying countermeasures can reduce the impact and the exploitability of events occurring.

Of course, the described approach for risk treatment is not applicable everywhere. It is not possible to make general statements, especially when certain activities are required and e.g. avoiding the risk would interfere with objectives. Sometimes the exploitability of the system under consideration is almost zero due to the effort required for a successful exploit, which despite a high impact may result in acceptance of a risk. Nonetheless, figure 6 presents an approach that is suitable in many cases.

Now that the risk assessment and the risk treatment process have been covered, all steps and their outcome need to be documented. The resulting reports can be used for future reference in order to trace back choices that were made. Furthermore, it can help in improving future risk management activities and when deciding on measures to take in future applications.

3.3.3 ISO/SAE DIS 21434 Based Risk Matrix

Also the ISO/SAE DIS 21434 (ISO/TC 22/SC 32, 2020) suggests to conduct risk management according to ISO 31000 (ISO/TC 262 Risk management, 2018). On top of that it provides propositions for 4x4 risk matrices with respect to the impact and exploitability that was derived from the risk assessment process. Moreover, it combines these values to retrieve severity values that support the decision making process when deciding on which risks to treat.

Concerning the exploitability of a certain risk the following values are possible (ISO/TC 22/SC 32, 2020):

- Very Low
- Low
- Medium
- High

For the impact the values below may be assigned (ISO/TC 22/SC 32, 2020):

- Negligible
- Moderate
- Major
- Severe

The ISO/SAE DIS 21434 does not enforce the utilization of fixed predefined severity values. However, it does define severity levels from 1 to 5 with 1 being the lowest and 5 being the highest possible value. This is due to the fact that individuals treat risk differently as risk management is a very subjective process. Also different application domains may be considered more or less critical which may affect the resulting risk matrix. An example of such a risk matrix is the symmetric one that is shown in table 1.

		Exploitability			
		Very Low	Low	Medium	High
Impact	Severe	1	3	4	5
	Major	1	2	3	4
	Moderate	1	2	2	3
	Negligible	1	1	1	1

Table 1. A symmetric risk matrix according to ISO/SAE DIS 21434 (ISO/TC 22/SC 32, 2020)

Although the threat modelling approach utilizes this matrix and predefines the matrix with these exact values, they are freely configurable to support company and application specific risk management approaches. A different configuration of this matrix will be discussed in section 4.3.4.

Now that the overall procedure of the threat modelling process has been discussed and demonstrated with an illustrative example and the concept of risk management has been explained, it is necessary to define the building blocks of the threat model. In order to understand how the system is analyzed, and where the displayed threats are coming from, the concept of rules demands explanation.

3.4 Rule-Based Approach

Rules represent the core of the threat model. They are defined as anti-patterns that should not occur in the system model. More precisely, this approach is based on (anti-)pattern matching. The rules currently describe generic configuration errors that might lead to exploits and are based on known weaknesses. When an analysis is conducted, the system model is checked against the threat model. Once there is a match, the applicable rule fires, depending on the security assumptions that were made when designing the system model.

A rule consists of:

- a **name** which resembles a title or short description of the threat or vulnerability
- a **description** which is a textual representation of the threat or vulnerability that it suffers from, also including high level mitigation strategies
- a **STRIDE category** to classify the type of threat
- an **impact value** depicting the severity or consequences of an exploit (FIRST, 2020b)
- an **exploitability value** depicting the ease of exploiting a vulnerability in terms of technical means or attack surface (FIRST, 2020b)
- a **rule text** which represents the anti-pattern in textual form

The following subsections provide a deeper insight into the contents of a rule. Firstly, STRIDE - a brainstorming method to identify and categorize threats - will be referred to. This will be followed by some examples of rules.

3.4.1 STRIDE

Many threat modelling approaches rely on STRIDE (Torr, 2005; Abomhara, Køien, & Gerdes, 2015; Hussain, Kamal, Rasool, & Iqbal, 2014; Desmet, Jacobs, Piessens, & Joosen, 2005; Shevchenko, 2018). STRIDE extends the CIA (Confidentiality, Integrity, Availability) model and correlates threats with security attributes (Lautenbach & Islam, 2016), thus yielding a view on threats from the attacker's perspective. By providing a brainstorming methodology based on six different categories it helps in identifying potential threats posed to a system.

Its goal is to make it easier to find possible attacks. Therefore, a component or component configuration is analyzed by each category to identify those threats. In the case of AIT's threat modelling approach the brainstorming has been conducted beforehand and the outcome has then been fed into the threat model. Consequently, STRIDE is used here to classify threats within six distinct categories. However, it is sometimes not feasible to provide an exact categorization as a threat may belong to multiple STRIDE classes. These categories utilized during brainstorming and the consequent categorizations are (Shostack, 2014; Hamad, 2020):

Spoofing means pretending to be someone else. It violates authentication, e.g. IP Spoofing.

Tampering is referred to when an entity modifies data, data flows or processes. It messes with the integrity of the data e.g. when someone adds/removes packets over a network, or changes contents of data.

Repudiation means claiming to be irresponsible for a certain event. It violates the ”non-repudiation” property of a system.

Information Disclosure describes getting access to data without proper authorization. This affects the confidentiality of the data.

Denial of Service threats tend to consume all the resources of services and hinder actual traffic from reaching its target. A common example is establishing a very high amount of connections to either cause the service to crash, or to block legitimate communication. A denial-of-service attack messes with the availability of a system.

Elevation of Privilege describes threats that exploit vulnerabilities within a system allowing a user to execute code without authorization or as admin.

STRIDE is only utilized for enumerating possible flaws of a system. It does not cover mitigation mechanisms or attack patterns.
After talking about STRIDE and the identification of potential threats within a system, the next section will show an excerpt of rules regarding the system and threat model from the examples before.

3.4.2 Examples of Rules

As already specified, rules fire when an anti-pattern is detected, more precisely, when there is a configuration in the system model that is considered exploitable. This is also the case for the system model from before, where a threat catalogue was derived which served as input for the subsequent risk treatment. In order to make clear why some of the threats were detected, the threats that were treated in the example introduced in section 3.2.3 will be explained.

In terms of the *distributed denial-of-service attack* threat, the according rule looks like the following:

```
Type(”Server”).tv(DDoS Mitigation != YES)
```

The above rule checks for an element of type server contained in the diagram which has a *DDoS Mitigation* that is not ”yes”. ”tv” stands for tagged value and depicts an attribute. It is relevant to specify that is is not ”yes” rather than ”no” due to the fact that the value could also be ”undefined”. As it may simply not have been specified, it is of importance to also include possibly not considered values within the analysis. As specified in section 3.2.3 applying a *DDos Mitigation* can eliminate this threat.

The second result that was revealed in the threat catalogue and then underwent risk treatment was the *IP Spoofing* threat whose rule text is displayed below:

```
Type(”ANY”).hasConnector(Connector.from(Type(”ANY”))
.tv(Authentication != YES))
```

This rule specifies any element that has a connector to any other element where no *Authentication* is specified. This basically means that anybody may access the provided service while they could tell the service to be someone else. This threat was mitigated by adding authentication e.g. in the form of login credentials, cookies containing session data or json web tokens.

Finally, the rule regarding *SQL Injection* will be shown:

```
Type(”ANY”).tv(Input Sanitization != YES)
.hasConnector(Connector.to(Type(”Database Server”)))
```

It searches for any element that has no *Input Sanitization* and a connector to an element of type database server. This structure basically describes an entity, most likely a server that does not sanitize the input before forwarding it to the database which might result in a possible *SQL Injection.* This is due to the fact that unsanitized data may execute unwanted queries on the database. Therefore, all queries should be checked and only allowed statements should be executed. By adding *Input Sanitization* this threat can be mitigated.

An exact specification of the grammar underlying these rules can be found in the grammar documentation[1]. Of course, also other, more sophisticated rules can be formulated utilizing this grammar. E.g. multiple attributes could be defined and combined with a logical "and"/"or". However, these few examples should provide sufficient insight into the capabilities of such rules. Moreover, for the approach envisaged in this master thesis only "type" and "tv" are relevant to identify specific products.

This section dealt with examples for rules with regard to the illustrative system model. In the following, shortcomings of the current threat model and possible improvements will be discussed.

3.5 Improving Threat Modelling at AIT

Although the current approach of integrating common configuration errors and generic threats into an analysis yields an improved design phase and can contribute to the process of making systems more resilient, it is not sufficient for real-world applications with running systems that are exposed to real-world threats. This generic approach does not include threats introduced through software updates or zero-day vulnerabilities. It helps in designing secure systems with best practice information but without regard to real threats. This approach is suited for the design phase but not for the operational phase. It is quite limited due to the fact that many components have security flaws even though they should theoretically be safe. To improve this, vulnerability data from online repositories can be used to derive information concerning specific software or hardware products.

3.5.1 Real-World Vulnerability Information

The CVE (MITRE, 2020a) and the CPE (Cheikes, Waltermire, & Scarfone, 2011) represent a promising source of information. From these it is possible to derive vulnerable products, the version that is vulnerable, as well as the specific vulnerability. On top of that, the NVD (NIST, 2020d) adds more information by assigning a score to the vulnerabilities and providing references to advisories that may help fixing the vulnerabilities.

The information that can be used to improve the currently generic threat model is shown in table 2.
By utilizing this additional information regarding specific products, it is feasible to analyze real-world threats. Therefore, it is possible to include known vulnerabilities of existing products and consequently, to deduce threats that are applicable.

The product alongside its version can be used to identify specific hardware or

[1] `https://documentation.threatget.com/20.09/Administration/Rules.html#full-specification-of-the-language`

Product	The exact product that is vulnerable
Version	The exact versions of the product that are vulnerable
Vulnerability	The description of the vulnerability that the product suffers from
CVSS	The vulnerability score assigned by NVD
References	A reference to advisories and solutions
Precondition	A description of prerequisites for compromising the system
Postcondition	A description of an attacker's possibilities after compromising the system

Table 2. Usable information from an NVD entry

software components. These can be linked with vulnerability, CVSS, references, pre- and postconditions. Adding a product to the graphical model and specifying the version attribute enables the automatic detection of vulnerabilities that are associated with the specific product.

Moreover, the CVSS enables the assignment of a severity score to each vulnerability. Comparing the scores of various threats can support decision making when deciding on which risks to treat. It also gives an insight into the impact and the exploitability of a vulnerability (FIRST, 2020b). The references that the NVD provides describe possible solutions to vulnerabilities for specific versions of software or hardware. They provide patches that can be applied to components in order to fix vulnerabilities or supply advisories on how to securely utilize the components.

The pre- and postcondition are not included in all vulnerability descriptions. Nonetheless, when an attacker exploits a vulnerability, the postcondition delineates what they can do after compromising the system. Furthermore, a precondition tells us conditions that are required in order to compromise a system. This yields additional reasoning on risk treatment.

Integrating all this additional information into the threat model and the threat analysis process can leverage awareness of currently unknown risks of system components. Moreover, it can inform decision on mitigation strategies to be used.

3.5.2 Adding more Level of Detail

During the course of the previous sections, the threat modelling process was explained. However, to make our system more resilient towards real-world threats, it is necessary to integrate additional up-to-date information into the generic threat model to make it applicable for actual known threats.

The current process is very well suited for the concept phase of a product. Usually system developers start with a high level system design before deciding on the exact system components to integrate. This is something that can already be covered with the existing approach of integrating generic threats into the system and simply defining whether or not e.g. communication in a wireless network is encrypted.

The method described above can give a good overview of the threat landscape in terms of which measures needs to be considered when developing a technological system. Although it enables the creation of a high level design, it does not consider flaws in software or hardware that are integrated into a system. However, it is simply not possible to know about all the vulnerabilities that products suffer from. And in the same way developers cannot be familiar with all the attack patterns used by adversaries. Technology is evolving. The result is an increasing amount of attack types.

Although a high level system design represents a first step into application security, more level of detail is required. Only knowing that unencrypted communication across trust boundaries is insecure is not sufficient. When implementing a technological system it is important to know what real weaknesses and vulnerabilities there are, and which threats exist to exploit them. Consequently, specific information on components presents a vital contribution to the threat analysis process.

The more that is known about a system, the more effective the result of the threat modelling process will be (Torr, 2005). This enables the search for specific vulnerabilities within system components.

The sharing of information about threats and vulnerabilities enables experts to react faster to incidents. However, the time passing between the disclosure of a vulnerability and the application of a patch is time in which the system is vulnerable. To keep systems resilient, it is therefore of great importance to be aware of the newest potential security risks (Meland et al., 2014; Tounsi & Rais, 2017).

But adding all this information by hand, is a very inefficient process. Moreover, manually scanning through resources daily is time consuming. When considering the CVE/NVD with approximately 140.000 entries, this becomes impossible to maintain. This is why the goal of this master thesis is to automate this decisive step and connect to existing online repositories, retrieve their data and generate rules automatically utilizing the technologies discussed in the following section.

3.6 Enabling Technologies

In order to allow for an more time-efficient method to derive rules, various technologies were evaluated. The following sections describe the technologies that were selected for automatically integrating real-world information.

3.6.1 Vulnerability Databases

A vulnerability database represents efforts about security flaws within hardware and software. Due to the fact that gathering all this information provides a major challenge because of the vast amount of data, there are joint efforts to collect vulnerabilities from various sources and put them together in one single database. (Granova & Slaviero, 2014).

When the number of yearly reported security issues was located around 3.500 in the year 2000, this number increased to over 20.000 in 2017. This results from an increasing amount of software published, as well as a growing awareness of possible security risks. (Eiram, 2018)

One of the major roots for vulnerability databases is the Common Vulnerabilities and Exposures (CVE) which serves as a dictionary for enumerating vulnerabilities in order to uniquely identify them. It is fed with information from various software vendors, government agencies and vulnerability researchers (MITRE, 2019b). But CVE is more of an enumeration instead of a vulnerability database. It is important to consider the granularity as well as the amount of information associated with a vulnerability. The information provided in a single CVE entry is usually not enough and requires a lot of additional manual work.

Therefore, NIST provides the National Vulnerability Database (NVD). *"The NVD is the U.S. government repository of standards based vulnerability management data [...]. This data enables automation of vulnerability management, security measurement, and compliance. The NVD includes databases of security checklist references, security related software flaws, misconfigurations, product names, and impact metrics."* (NIST, 2020d). The NVD builds upon the CVE and further investigates CVE entries. It adds additional value by making the information within a CVE accessible in a structured manner and provides information on the exact products (CPE) used, while providing the weakness (CWE) as well as references for mitigation strategies. Moreover, the associated impact metrics (CVSS) can help with measuring security. The NVD is freely accessible. (MITRE, 2019a)

Another free resource for security information and vulnerability intelligence is Packetstorm. It promotes itself as *"important for the personal Internet user, corporations, and governments to stay aware of vulnerabilities that may affect their systems"* (Packetstorm, 2020). Furthermore, Packetstorm provides tools to support mitigation and publishes all information through RSS feeds, Facebook and Twitter.(Packetstorm, 2020)

Other providers, such as VulDB and VulnDB charge fees for accessing their data. Therefore, they offer more comprehensive information on vulnerabilities. These paid vulnerability aggregators cover additional tactical information that cannot be found in the original reports. Furthermore, they include more metrics to leverage reasoning in terms of risk severity. A list of existing vulnerability databases can be found in table 3.

Database	**Link**
CVE	`https://cve.mitre.org/`
NVD	`https://nvd.nist.gov/`
Packet Storm Security	`https://packetstormsecurity.com/`
Vulners	`https://vulners.com/`
VulDB	`https://vuldb.com/de/`
VulnDB	`https://vulndb.cyberriskanalytics.com/`
Snyk	`https://snyk.io/`
Exploit-DB	`https://www.exploit-db.com/`
Microsoft Security Bulletins	`https://docs.microsoft.com/en-us/security-updates/securitybulletins/securitybulletins`
Mozilla Foundation Security Advisories	`https://www.mozilla.org/en-US/security/advisories/`
Vulnerability Lab	`https://www.vulnerability-lab.com/`
0day today	`https://0day.today/`
Rapid7	`https://www.rapid7.com/db/`
Vulnerable Things (Pilot)	`https://vulnerablethings.com/`

Table 3. A list of vulnerability databases

This master thesis focuses on free alternatives and will include data from the NVD and Packetstorm. But before going into detail about that, technologies that the NVD builds upon require explanation. These technologies will be elaborated on in the following sections.

3.6.2 CPE - Common Platform Enumeration

The Common Platform Enumeration 2.3 represents standardized methods of assigning names to IT products. It can be used to uniquely describe applications, operating systems or hardware devices on the market and, consequently, helps in identifying products used within a company. Moreover, it supports the IT management policies of a company by relating these products or assets with vulnerability information, configuration data and remediation policies by uniquely identifying installed products. These standardized methods also enable automated decision making when finding one of these products within the work environment. (Cheikes et al., 2011)

The CPE supports three methods of representing an IT product. The following examples show these possible representations for the Microsoft Internet Explorer 8.0.60001 Beta (Cheikes et al., 2011):

1. The well-formed CPE name (WFN) is an *"abstract logical construction"*(Cheikes et al., 2011, p. 1). An example is given below:

   wfn:[part="a",vendor="microsoft",product="internet_explorer",
   version="8\.0\.6001",update="beta"]

2. But CPE also supports machine-readable encodings that bind to WFN. It allows for binding the WFN to a Uniform Resource Identifier (URI) which is mostly used for backward compatibility with CPE 2.2. The URI representation of the example from before looks like the following:

   cpe:/a:microsoft:internet_explorer:8.0.6001:beta

3. The binding that is utilized in CPE 2.3 is the representation as a formatted string or formatted string binding. It adds more information to a product by adding additional attributes. Below is an example of such a formatted string binding:

   cpe:2.3:a:microsoft:internet_explorer:8.0.6001:beta:*:*:*:*:*:*

As can be seen in the formatted string binding example above, some attributes are only marked with an asterisk ("*"). This is due to the reason that these attributes are not specified within this CPE string. In general a CPE entry consists of eleven attributes to be considered upon creation.

A CPE starts with a specification of the version. In this case, "cpe" and "2.3" indicate a CPE version 2.3. After this version specification, the specific attributes are assigned. These possible attributes are listed below (Cheikes et al., 2011):

- **part** - depicts whether a product is an *a: application, o: operating system or h: hardware device*
- **vendor** - depicts the creator/maintainer of the product
- **product** - represents the name of the product
- **version** - specifies the version of the product
- **update** - describes the update version of the product
- **edition** - specifies edition-related terms by the product vendor. It is used for CPE 2.2 backwards compatibility
- **language** - describes the language of the user interface, defined in RFC5646
- **software edition** - describes the specific edition of the software for certain markets
- **target software** - describes the targeted environment
- **target hardware** - describes the instruction set architecture. (e.g. x86, x64, Java Virtual Machine, Common Language Runtime, VMs)
- **other** - describes any other information that is not covered within the other attributes

In our case only the first five attributes are set. The attribute **part** with value *a* means that the product is an application. Moreover, **vendor** - which in this example is *microsoft* - shows the creator the respective product. The **product** *internet_explorer* as well as **version** *8.0.6001* give information on the product and its version. In case of this example the **update** attribute tells us that we are dealing with a *beta* version of the microsoft internet explorer.

Not all attributes contained in a CPE need to be included to form a valid entry. Some yield additional information, this is to say details about **target software** or **target hardware** which give more insight into products that may be aimed at cross-platform operation.

CPE provides compact and standardized methods of describing software and hardware in a unique way enabling the exact identification of a product within a system. This information can become useful when conducting threat modelling with real-world applications to detect related vulnerabilities. This is where the CVE, which contains a high amount of these related vulnerabilities, plays an important role. It will be discussed in the following section.

3.6.3 CVE - Common Vulnerabilities and Exposures

The Common Vulnerabilities and Exposures (CVE) has existed since 1999 and was launched by the MITRE Corporation. It is a community effort on an international basis with the aim of becoming a standardized basis for vulnerability exchange, tool evaluation and vulnerability referencing. Until the launch of CVE, companies had their own custom ways of reporting vulnerabilities which made it complicated to distinguish between vulnerabilities that were already covered and possibly untreated vulnerabilities. This led to many gaps in the system design and, consequently, caused many security issues.(MITRE, 2019b)

Since then the CVE has become the industry standard for vulnerability identification. It provides a basis for data exchange between cyber-security products and helps in identifying vulnerable products. Moreover, it can aid the decision for choosing tools for a company when vulnerabilities are publicly known. (MITRE, 2019b)

CVE *"is a list of common identifiers for publicly known cyber-security vulnerabilities"* (MITRE, 2019b). It is used to uniquely identify vulnerabilities and share information regarding vulnerabilities. Each CVE entry has a unique identifier, the CVE ID. This makes is possible to identify each vulnerability in systems distinctly. Furthermore, a CVE includes a standardized description for vulnerabilities. It can be seen as a dictionary that can be consulted with regard to its ID. Integrating CVE into a product raises the awareness of vulnerabilities and - if applied correctly - can lead to improved security. (MITRE, 2019b)

CVEs are created by so-called CVE Numbering Authorities (CNAs). When an entity discovers a potential vulnerability that they think is currently unreported they can send a request to a CNA. These CNAs can assign CVE IDs to vulnerabilities concerning specific products (MITRE, 2020b) and create a description alongside references. Afterwards the entry is added to the CVE list and is ready to be consumed by the community. Table 4 provides an example for a CVE entry.

CVE ID	CVE-2020-9586
Description	Adobe Character Animator versions 3.2 and earlier have a buffer overflow vulnerability. Successful exploitation could lead to arbitrary code execution.
References	https://helpx.adobe.com/security/products/character_animator/apsb20-25.html

Table 4. CVE Example based on CVE-2020-9586

The CVE is publicly available and free to use. The entries contained in the CVE dictionary contain promising information to be used in threat modelling. The information that can be obtained is with regard to real-world information and contains vulnerabilities of existing components.

3.6.4 CWE - Common Weakness Enumeration

The Common Weakness Enumeration (CWE) was launched in 2006 by the MITRE Corporation. It is a community effort to aid developers and designers to prevent common architectural flaws that might lead to vulnerabilities before product deployment. Its main goal is to identify weaknesses in products. *”“Weaknesses” are flaws, faults, bugs, vulnerabilities, or other errors in software or hardware implementation, code, design, or architecture that if left unaddressed could result in systems, networks, or hardware being vulnerable to attack.”*(MITRE, 2020c).

Originally developed for software weaknesses, the CWE was soon adapted to also support hardware weaknesses. CWE presents a common language for weaknesses. CWE entries are represented in a hierarchical structure in tree form (NIST, 2020c). This allows different levels of granularity. The CWE is abstract on the top level nodes but gets more specific and more detailed the deeper the tree is traversed.

When considering a buffer overflow or in this case the CWE-121: Stack-based Buffer Overflow, the hierarchical structure was derived from the CWE website's ”list of weaknesses” concerning software development[2] and is as follows:

[2] `https://cwe.mitre.org/data/definitions/699.html`

- **Software Development**
  - API / Function Errors
  - Audit / Logging Errors
  - ...
  - **Memory Buffer Errors**
    - Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
    - Write-what-where Condition
    - ...
    - **CWE-787: Out-of-bounds Write**
      - **CWE-121: Stack-based-Buffer Overflow**
      - CWE-122: Heap-based-Buffer Overflow
      - CWE-123: Write-what-where Condition
      - CWE-124: Buffer Underwrite ('Buffer Underflow')
      - ...

At the very top level, *Software Development* represents the CWE list that the weakness is located in. Below that are numerous child elements that are meant to divide CWEs into categories. In this case it is the *Memory Buffer Errors* category which contains the *CWE-787: Out-of-bounds Write*. To make this CWE more fine granular, various further child elements are attached (NIST, 2020c), such as the *CWE-121: Stack-based-Buffer Overflow* or the *CWE-122: Heap-based-Buffer Overflow*.

CWEs consist of a description and of relationships between parent and child elements (MITRE, 2020c) - every parent knows its children and every child knows its parent. But it also contains an abstract version of a flow of action. CWE entries also link to other CWEs that can follow when a specific weakness is exploited and - vice-versa - what could have happened before it was exploited. Many CWEs include examples for demonstration, e.g code examples that illustrate how a Stack-based Buffer Overflow might be enabled. Also detection methods such as the suggestion of using analysis tools are contained within a CWE (MITRE, 2020c). Furthermore, a CWE may include potential mitigation strategies for the different phases of product development. In case of the Stack-based Buffer Overflow mitigation during build and compilation phase it suggests:

"Run or compile the software using features or extensions that automatically provide a protection mechanism that mitigates or eliminates buffer overflows. For example, certain compilers and extensions provide automatic buffer overflow detection mechanisms that are built into the compiled code. Examples include the Microsoft Visual Studio /GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice." (MITRE, 2020d)

The information presented in a CWE is quite informative and can help in understanding and handling potential weaknesses of a technological system.

3.6.5 CVSS - Common Vulnerabilities Scoring System

The Common Vulnerabilities Scoring System (CVSS) was launched in 2005 and is currently managed by the Forum of Incident Response and Security Teams (FIRST). It is an *"open framework for communicating the characteristics and severity of software vulnerabilities"* (FIRST, 2020b, p. 1). Its main objective is to assign a criticality score between 0 and 10 to a vulnerability by considering different metrics.

The CVSS comprises three groups regarding the metrics (FIRST, 2020b):

1. The **base score** which usually stays unchanged after assignment and stays the same in all various environments. It is a score that works agnostic of its environment.
2. The **temporal score** which can be assigned when the criticality of a certain vulnerability changes over time. This can be due to advanced exploit mechanisms or patches.
3. The **environmental score** includes the characteristics of the system environment to refine the criticality.

The NVD focuses on the CVSS base score. Therefore, this master thesis will also focus on the capabilities of the base score and utilize its means of vulnerability scoring.

The CVSS base score is used in vulnerability and, consequently, in threat management. It represents an important part of decision making when determining which risks to treat and how to treat them. It can provide help in ranking threats within an organization's infrastructure. (FIRST, 2020b)

A CVSS score consists of two sub-metrics. On the one hand, it comprises the exploitability metrics which describe the difficulty and the efforts required in reaching an exploit. It incorporates the properties that need to be fulfilled in order to start a successful attack. On the other hand, impact metrics help in identifying the consequences of a potential exploit and the effect on a component (FIRST, 2020b). These metrics will be discussed in the following.

The exploitability metrics are comprised of (FIRST, 2020b):

1. **Attack Vector (AV)**, which describes the location of the potential attack. The nearer the adversary needs to be to the target the lower the score. If an attack can be conducted over the network, the value will be larger than an attack that requires physical interaction. This is due to a larger attack surface, as adversaries over the network represent a much larger group than insiders.

2. **Attack Complexity (AC)**, which deals with properties that cannot be directly controlled by the attacker and may depend on additional knowledge about the attack target. The higher the amount of knowledge required, the lower the final CVSS score actually is.
3. **Privileges Required (PR)**, which represents a metric that tells if an adversary requires additional privileges in order to conduct an exploit.
4. **User Interaction (UI)**, which describes whether an attack can be conducted directly or if a possible exploit requires additional user input.

The exploitability metrics represent the effort and the technical means to conduct successful attacks, while the impact metrics are made up of (FIRST, 2020b):

1. **Confidentiality (C)**, which represents the impact on the confidentiality of information when an exploit happens. It deals with possibly missing access restrictions and information disclosure.
2. **Integrity (I)**, which represents the impact on the integrity of data. *"Integrity refers to the trustworthiness and veracity of information"* (FIRST, 2020b, p. 10)
3. **Availability (A)**, which represents the effects on the availability of a component. An attack targeting availability results in reduced availability or total unavailability. It usually refers to a service and whether its resources are accessible. Attackers tend to consume the bandwidth or the processing time of the service.

All these metrics have numbers assigned to them depending on the severity level of the observed metric. The respective values can be found in the CVSS 3.1 specification document and the ones regarding the base score are listed in table 5.

When looking at table 5 in detail, one will notice that the value of "Privileges Required" is affected by an attribute called scope. The additional scope attribute, which is also included inside a CVSS score but not directly considered a metric, deals with taking different security scopes into account. More precisely, a compromised component can influence another component within a different security scope which could also represent a third party.

A CVSS score is created by assigning a value to each of these metrics. The resulting CVSS is displayed in the form of a vector string which is a representation of the metrics with their assigned values in textual form. An example for a base score in vector string format is shown below[3]:

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

The first part before the slash denotes the CVSS version which in this case is 3.1. The remaining slices between the slashes represent the metric with the assigned values. E.g. *AV:N*, which represents the **Attack Vector** with an assigned

[3] `https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N`

Metric	**Metric Value**	**Numerical Value**
Attack Vector	Network (N)	0.85
	Adjacent (A)	0.62
	Local (L)	0.55
	Physical (P)	0.2
Attack Complexity	Low (L)	0.77
	High (H)	0.44
Privileges Required	None (N)	0.85
	Low (L)	0.62 if scope unchanged, else 0.68
	High (H)	0.27 if scope unchanged, else 0.5
User Interaction	None (N)	0.85
	Required (R)	0.62
Confidentiality,	High (H)	0.56
Integrity,	Low (L)	0.22
Availability	None (N)	0.0

Table 5. CVSS base score metric values taken from (FIRST, 2020b)

value of **Network** or *AC:L* which represents the **Attack Complexity** with an assigned value of **Low**. All the slices that are contained inside a vector string are represented in table 5.

After assigning values to the metrics, two equations, one that calculates the exploitability and one that calculates the impact are combined to receive the final base score. The resulting numerical value then gives an insight into the criticality of a vulnerability. The vector string from before has the following values associated:

- **AttackVector (AV)** - *Network* - 0.85
- **AttackComplexity (AC)** - *Low* - 0.77
- **PrivilegesRequired (PR)** - *None* - 0.85
- **UserInteraction (UI)** - *None* - 0.85
- **Scope** - *Unchanged*
- **Confidentiality (C)** - *High* - 0.56
- **Integrity (I)** - *High* - 0.56
- **Availability (A)** - *None* - 0.0

The calculation[4] for the exploitability works as follows (FIRST, 2020b):

$$\textbf{Exploitability} = 8.22 * AV * AC * PR * UI \tag{1}$$

This results in:

$$\textbf{Exploitability} = 8.22 * 0.85 * 0.77 * 0.85 * 0.85 \approx \textbf{3.887} \tag{2}$$

[4] Please note that the abbreviated version of the values from above is utilized within the formulas

The calculation for the impact starts with calculating the impact sub-score (ISS) (FIRST, 2020b):

$$\textbf{ISS} = 1 - [(1 - C) * (1 - I) * (1 - A)] \tag{3}$$

$$\textbf{ISS} = 1 - [(1 - 0.56) * (1 - 0.56) * (1 - 0.0)] = \textbf{0.8064} \tag{4}$$

Afterwards the impact score is calculated depending on whether the scope is changed or not. If the scope is unchanged, as in our case, the calculation is as follows:

$$\textbf{Impact} = 6.42 * ISS \tag{5}$$

$$\textbf{Impact} = 6.42 * 0.8064 \approx \textbf{5.117} \tag{6}$$

If the scope is changed the calculation requires the following equation:

$$\mathit{Impact} = 7.52 * (ISS - 0.029) - 3.25 * (ISS - 0.02)^{15} \tag{5a}$$

By combining these equations the base score can be derived (FIRST, 2020b).

If the impact is 0 or smaller:

$$BaseScore = 0 \tag{7}$$

If the scope is unchanged as in our case the formula below is used:

$$\textbf{Base Score} = \mathit{Roundup}(Minimum[(\mathit{Impact} + \mathit{Exploitability}), 10]) \tag{8}$$

$$\textbf{Base Score} = \mathit{Roundup}(Minimum[(5.117 + 3.887), 10]) = \textbf{9.1} \tag{9}$$

If the scope is changed another formula applies:

$$BaseScore = \mathit{Roundup}(Minimum[1.08 * (\mathit{Impact} + \mathit{Exploitability}), 10]) \tag{8a}$$

The calculation for the base score in this example yields a 9.1 which is highly critical, as severity values range from 0 to 10. Additionally, the CVSS 3.1 specification includes a Rating scale to divide scores into different categories depending on the value of the score.

Table 6 shows the severity ratings for the ranges of a CVSS score.

CVSS can play an important part in the risk management of the application

Rating	CVSS Score
None	0.0
Low	0.1 - 3.9
Medium	4.0 - 6.9
High	7.0 - 8.9
Critical	9.0 - 10.0

Table 6. Severity Rating (FIRST, 2020b)

of threat modelling. The exploitability metric of the score will be mapped as likelihood inside the risk matrix, while the result from the impact equation can represent the impact inside the matrix as well as provide aid with the STRIDE classification for rules. Thus, by conducting an appropriate mapping, the CVSS can enhance risk management by utilizing exploitability and impact metrics to derive the severity score for the risk matrix. Moreover, the CVSS base score itself can also represent an important part of the decision making process. In addition, applying parts of the CVSS to threat modelling, can enhance rule creation and enable automated severity scores and STRIDE classification.

3.6.6 NER - Named Entity Recognition

Named Entity Recognition represents a fundamental task within the topic of Natural Language Processing (NLP) (Lei et al., 2014) and is denoted as the *"starting point for most information extraction applications"*(Jurafsky & Martin, 2000, p. 761). The purpose of Named Entity Recognition is to detect entities within a text and to classify them according to predefined classes. There are various applications that utilize Named Entity Recognition in their information extraction tasks. They range from question answering systems, machine translation systems, summarization systems (Goyal, Kumar, & Gupta, 2017) to news-oriented information extraction systems where focus is put on detecting people, locations and organizations.

A listing of possible classes and examples for a news-oriented environment is presented in Table 7.
It features very simple examples of Named Entity Recognition where only one label per sentence is assigned. But this example still gives a good insight into an application of Named Entity Recognition. Named entities are not always just single words, they can also consist of multiple constituent words, such as *Mt. Sanitas* and *Mini Cooper* in table 7. Two capitalized words may represent a name and may be an indicator of a named entity. But also other words could be an indicator: E.g. the degree of *Dr.* might indicate a possible categorization (Jurafsky & Martin, 2000) for the following capitalized words inside the **People** class. Or in a more modern example a *BSc* or *MSc* following two capitalized words may indicate **People**. Even numbers might represent a possible date, money or price entity.

Class	**Example**
People	*Turing* is often considered to be the father of modern computer science.
Organization	The *IPCC* said it is likely that future tropical cyclones will become more intense.
Location	The *Mt. Sanitas* loop hike begins at the base of Sunshine Canyon.
Vehicle	The updated *Mini Cooper* retains its charm and agility.

Table 7. Named entity classes and examples (Jurafsky & Martin, 2000, p. 762)

Text is often a source of ambiguity. One and the same expression may be subject to various semantic interpretations. A proper name, for example, may well take different meanings. Considering the term of *JFK* different named entity classes could be assigned, as JFK could be the former president of the USA but also an airport (Jurafsky & Martin, 2000). Obviously, entities may belong to various categories.

Another example of ambiguities based on the word *Washington* is given by Jurafsky and Martin:

1. [**People** *Washington*] was born into slavery on the farm of James Burroughs.
2. [**Organization** *Washington*] went up 2 games to 1 in the four-game series.
3. Blair arrived in [**Location** *Washington*] for what may well be his last state visit.

As a matter of fact, *Washington* can have various meanings. In the cases presented here, it could be president *Washington*, an organization (in this case a sports team) or a location such as the city named *Washington*. Multiple labels seem suitable for the word *Washington*. Therefore, looking only at the words as such is not sufficient and requires more context.

Named Entity Recognition approaches this problem with a sequence labelling task (Che, Wang, Manning, & Liu, 2013). Sentences are tagged word-by-word. A common approach to this is the **IOB** scheme. Where **I** denotes that a word is part of a named entity, **B** describes the beginning of a named entity and **O** represents words that are outside of named entities (Jurafsky & Martin, 2000). The approach of sequence labelling serves the purpose of training classifiers to label words with tags that may indicate a named entity.

When selecting the features utilized during training one should be careful that these features are reliable predictors for the classes to be assigned. These features do not necessarily have to be contained within the words to be classified. They may also be a representation of the context that named entities are surrounded by, such as predictive words or n-grams, which are combinations of words that indicate the occurrence of a named entity. Furthermore, the parts-of-speech or the orthographic pattern of a word may also influence the detection of a named entity. (Jurafsky & Martin, 2000)

When developing a Named Entity Recognition System, the selected features present the most important part of a successful system. Figure 7 shows the general approach to creating a Named Entity Recognizer.

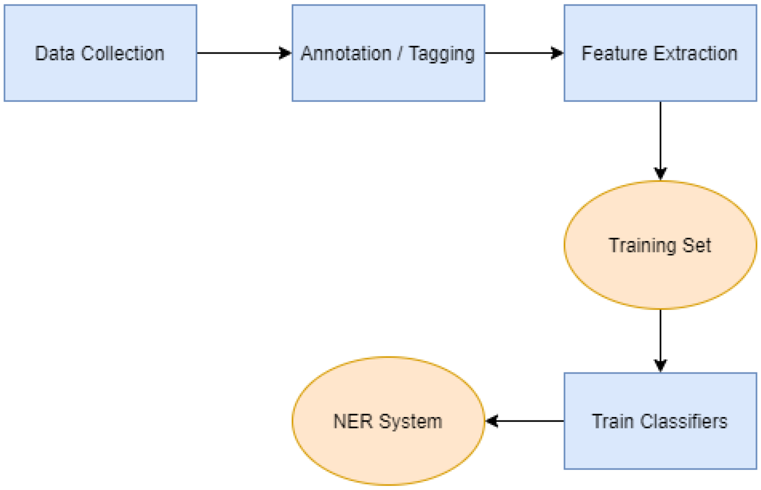


Fig. 7. Building an NER System based on (Jurafsky & Martin, 2000, p. 762)

As the first step, documents that fit the use case need to be collected. Afterwards these documents are annotated with labels that represent the categories of the data as best as possible. These documents then undergo a feature extraction step which outputs a training set which is suitable for training sequential classifiers used within the application. Once training has been conducted the NER System is ready to be tested and used. (Jurafsky & Martin, 2000)

The evaluation of a Named Entity Recognizer is done with precision, recall and the F_1 measure. Therefore we need to define the measures used inside the calculation (Bird et al., 2009):

1. **True Positives** (TP) are entities that are correctly considered relevant.
2. ***True Negatives*** *are not used in the formulas regarding precision and recall because simply marking everything as negative would distort the results and even ill-trained NER systems could achieve good results.*
3. **False Positives** (FP) are entities that are incorrectly considered relevant.

4. **False Negatives** (FN) are entities that are incorrectly considered irrelevant.

These measures are utilized in the following metrics for calculating the performance of the Named Entity Recognition System (Bird et al., 2009):

Precision is a metric that compares the number of correctly classified entities to the total number of classified named entities, including incorrectly classified ones.

$$Precision = \frac{TP}{(TP + FP)} \tag{10}$$

Recall represents the ratio of correctly classified entities and the total number of entities that should have been classified.

$$Recall = \frac{TP}{(TP + FN)} \tag{11}$$

F$_1$ measure denotes a way of combining precision and recall within a single score. It is their harmonic mean. The result is in the range of 0 to 1, where a value close to 1 points out that the system is performing well while a value close to 0 means poor performance.

$$F_1 measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{12}$$

Many applications focus on the extraction of people, locations and organizations, which is why many libraries (Bird et al., 2009; Finkel et al., 2005; Honnibal & Montani, 2017) offer pre-trained solutions that already offer these classes. When it comes to the security sector, however, there are - to the best of our knowledge and due to intense research - no pre-trained entity extractors contained within these libraries. Only two external solution regarding cyber domain entities also targeting CVE were found (*stucco/entity-extractor*, 2020; Mariani, 2020). However, (*stucco/entity-extractor*, 2020) did not feature pre- and postcondition and (Mariani, 2020) was focused on Linux Kernel specific entries.

4 Model and Method

Section 3 provided an overview of the current capabilities of the threat modelling approach at AIT and discussed technologies that the improved approach envisaged in this master thesis will build upon. On top of this basis, an outline of the affected steps of the threat modelling process as well as a high-level description of the necessary components for automatically improving the threat model will be given. Furthermore, a generic model for representing vulnerabilities from arbitrary sources will be introduced. This will be followed by a description of methods for mapping between the attributes from the NVD and the vulnerability model. Finally, the factors that influenced the model and the methods will be discussed.

4.1 Enhancements to Threat Modelling Persued at AIT

This section provides an overview of the current threat modelling approach and elaborates on possible improvements to be made. As discussed in section 3 the goal of this master thesis is to enhance the current generic threat modelling process with data from real-world applications. Although high-level information is very well suited for the concept phase, it is not sufficient when taking the whole system lifecycle into account.

Threats evolve over time, new vulnerabilities are detected within existing applications that may have been previously unnoticed. This takes effect especially when third party components are integrated into the system under consideration. Knowing about the exact vulnerabilities of hardware or software increases the precision of threat analysis results and yields additional information on the components to be integrated. Figure 8 gives an overview of the overall threat modelling process at AIT and shows where improvements can be implemented.

As already explained in section 3 there are two models utilized within the threat modelling approach:

Firstly, there is the system model which is a representation of the system under consideration and contains all security assumptions relevant to the system. It is a reflection of the system and its respective system components, communication channels and trust boundaries, as designed in reality.

Secondly, the threat model is currently a representation of generic threats and weaknesses that is capable of pointing out configuration errors. This is achieved with a rule-based approach where the rules are anti-patterns, this is to say patterns that a system should not contain. The threat model represents an extensible knowledge base, which can be extended by creating new rules.

Once threat model and system model have been developed thoroughly, the threat analysis step is conducted which compares the two models with each other, applies the rules to the system model and detects threats regarding the system configuration. The resulting threat analysis report then reveals threats posed to the system.

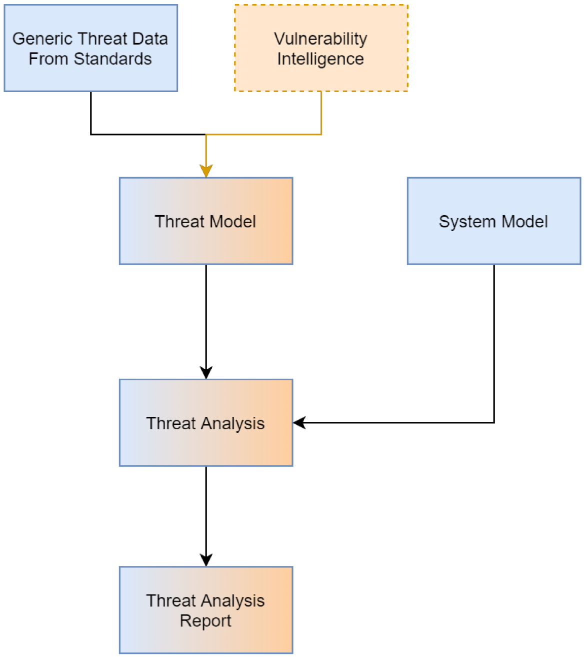


Fig. 8. General Overview of Threat Modelling at AIT

So far only the **blue** parts of figure 8 have been explained, as they represent the state of the art of the threat modelling approach. The **orange** parts of the diagram represent the newly introduced content. The general approach to threat modelling will remain untouched, but additional information from online repositories shall be included to enhance its capabilities. Moreover, threat intelligence or rather its sub-topic, this is to say vulnerability intelligence, will be introduced into threat modelling.

Therefore, vulnerability and exploit descriptions from online sources such as the CVE/NVD and Packetstorm shall be added to the threat model. But not only the textual description shall be included. The affected product, including the vulnerable versions as well as the specified weaknesses shall be extracted. This information will afterwards be used to create rules for specific components and real-world applications.

Apart from the vulnerability description, a link to the CWE will be added where it is feasible. Moreover, the CVSS will be used to automatically derive an impact and exploitability score for the risk matrix in order to deduce a severity score relevant for risk analysis and risk treatment and help with the STRIDE classification of the rules. Additionally, the original CVSS base score shall not be changed and still be available for further reference.

The generated information will impact all sectors of the threat modelling approach that are presented in figure 8 as gradient colors from **blue** to **orange**. The threat model will be enhanced with new rules, which can then be integrated into the threat analysis process to enable a more realistic threat analysis. A high-level overview of the approach achieving this goal is given in the following section.

4.2 High-Level Description of the Applied Approach

In order to supply the existing threat model with up-to-date information from online resources, it is necessary to automate the generation of rules. Figure 9 gives an overview of the underlying approach.

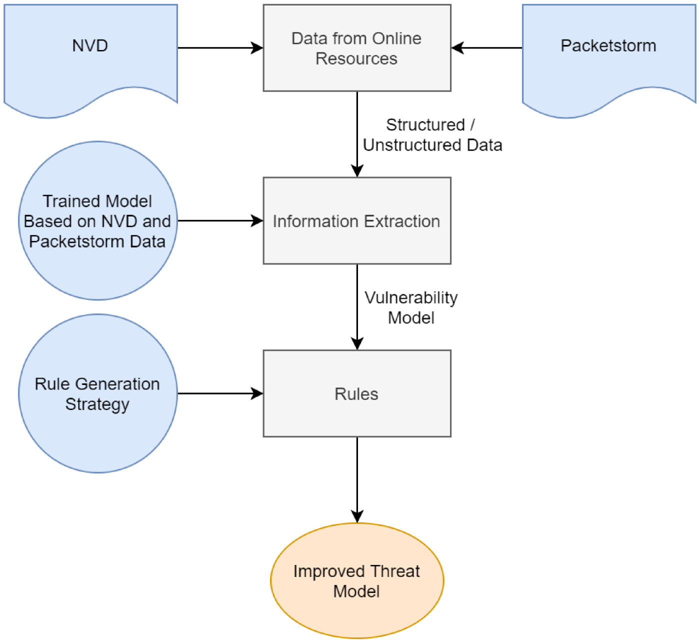


Fig. 9. Illustration of the applied approach

First of all, data from online resources, particularly from the NVD and Packetstorm must be retrieved. This information can come in structured as well as unstructured form. Therefore, an information extraction task is applied. It parses the content of a structured JSON file and utilizes an underlying Named Entity Recognition model that was trained on NVD and Packetstorm data to extract information from unstructured plain text contained within descriptions. This way a textual description of a vulnerability is processed and put into a structured format which is further enhanced with additional features coming from the

JSON file, such as vulnerability score, references and associated weaknesses. The resulting format is the vulnerability model which will be discussed in section 4.3.

From this model it is feasible to derive rules. These rules are based on component specific information and, thus, allow for an exact identification of vulnerable products. Real-world information in the form of rules leverages the capabilities of threat modelling and leads to an improved threat model.

Consequently, applying the improved threat model inside a threat analysis enables the detection of real-world vulnerabilities that existing products suffer from. The resulting threat analysis report utilizes the scoring and the detected vulnerabilities to improve the risk treatment process. The model behind this will be discussed in the following sections.

4.3 Representing Vulnerabilities

The vulnerability intelligence data gathered will be used to generate new rules with real-world vulnerability information. But different repositories provide this data in different formats. Some utilize structured formats that are XML or JSON based. Other platforms provide their information in an unstructured format that requires further analysis and cannot be directly integrated into the system. The following sections will explain the derivation of a common format, an adapter-based approach for joining information from various sources and the transformation into this common format. Afterwards, the extraction process of vulnerability related information from unstructured content will be elaborated on.

4.3.1 Deriving the Requirements for a Vulnerability Model

First of all, a common format needs to be defined, which makes it possible to unite the information from different sources into one common scheme that allows for the static generation of rules. In order to represent information on real vulnerabilities, this format must include:

1. The **product** that is vulnerable
2. The vulnerable **versions** of the product
3. The **vulnerability** that the product suffers from

These three attributes are the minimum requirements for creating rules that resemble actual vulnerabilities. They also represent the minimum information needed to identify a vulnerability within a system that utilizes third party components.

This information is contained in the following example taken from the exploits posted on Packetstorm[5]:

[5] `https://packetstormsecurity.com/files/tags/exploit/`

$\underbrace{\text{Impress CMS}}_{\text{Product}}$ $\underbrace{\text{version 1.4.0}}_{\text{Version}}$ suffers from a $\underbrace{\text{cross site scripting vulnerability}}_{\text{Vulnerability}}$.

As shown above, the three required attributes are contained within the text. *Impress CMS* represents the **product** name, *version 1.4.0* gives information about the affected **version** of the software and *cross site scripting vulnerability* depicts the **vulnerability**. In the case of Packetstorm, the information presented is unstructured and requires further analysis as it cannot be directly ingested and converted into a common format. However, this can be done with Named Entity Recognition which was discussed in section 3. The exact process of retrieving the data, pre-processing, creating a training set and the training of the classifiers will be discussed in section 5.

The next example has been taken from the CVE-2020-9586[6]:

$\underbrace{\text{Adobe Character Animator}}_{\text{Product}}$ $\underbrace{\text{versions 3.2 and earlier}}_{\text{Version}}$ have a
$\underbrace{\text{buffer overflow vulnerability}}_{\text{Vulnerability}}$. Successful exploitation could lead to
$\underbrace{\text{arbitrary code execution}}_{\text{Postcondition}}$.

The above sentences also contains the three attributes that are required to form a rule. *Adobe Character Animator* as the **product**, *version 3.2 and earlier*, represents the **version**. *Buffer overflow vulnerability* denotes the **vulnerability** that the CVE entry describes. Moreover, this entry contains a **postcondition**, this is to say actions that a potential adversary can take once the vulnerability has been exploited. In this case, a successful exploit of a *buffer overflow vulnerability* enables the **postcondition** *arbitrary code execution*. As not all CVE entries contain **postconditions**, this attribute should be optional. The same constraint is also relevant for a **precondition**.

When talking about the CVE, also the NVD deserves mentioning. It holds additional information on CVE entries and provides them within a REST API that returns data in JSON format. The data contained within the NVD entry of the previous CVE entry is shown in table 8.

The information that can be retrieved from the NVD is static and already

[6] `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-9586`

Attribute	**Example**
CVE ID	CVE-2020-9586
Description	Adobe Character Animator versions 3.2 and earlier have a buffer overflow vulnerability. Successful exploitation could lead to arbitrary code execution.
CVSS Base Score	7.8
CVSS Vector String	CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H
References	https://helpx.adobe.com/security/products/character_animator/apsb20-25.html
CWE	CWE-120: Buffer Copy without Checking Size of Input (’Classic Buffer Overflow’)
Affected Products	cpe:2.3:a:adobe:character_animator:*:*:*:*:*:*:*:*
Version	Up to (including) 3.2

Table 8. NVD Entry based on CVE-2020-9586

provides most of the unstructured textual description of a CVE entry within a structured format. The affected product is represented in a CPE string that allows for automatic extraction of the product name without the requirement of further text analysis. Furthermore, a CPE entry also contains the affected version of a product. When taking a look at the CWE attribute, the weakness that the product suffers from is also included. Consequently, the minimum requirements for rule creation are fulfilled with this simple extraction step.

But the NVD contains more information that could improve threat analysis. The CVSS base score, as well as the CVSS vector string can improve the risk treatment process and the references provide aid when considering countermeasures.

All this information can be extracted automatically without the need for Natural Language Processing. The only information that cannot be retrieved from CVE/NVD without further analysis are the pre- and postcondition which require Named Entity Recognition to be detected. Both attributes could be used for generating attack graphs (Lallie, Debattista, & Bal, 2020; Ou, Boyer, & McQueen, 2006; Jha, Sheyner, & Wing, 2002), which will not be included inside this master thesis but are well worth future research. Nevertheless, when considering other sources such as Packetstorm the information is presented in plain text which requires additional steps, such as the Named Entity Recognition.

Before defining a vulnerability model, the requirements need to be specified. These requirements are listed below:

1. It shall be possible to **integrate data from various sources**
2. The model shall be capable of **representing vulnerabilities**
   (a) The **affected product(s)** as well as **affected versions**
   (b) The **weakness** that an existing system suffers from

(c) The **description** of the vulnerability in textual form

3. The model shall contain a **score** (if available) to enable proper risk treatment
4. The model shall include **references** to advisories and solutions (if available)

The discussed requirements represent additional information that are being integrated into the threat model, leveraging the capabilities of the threat modelling process. Section 4.3.2 discusses the vulnerability model based on the derived requirements.

4.3.2 A Model for Automated Rule Generation from Structured and Unstructured Data Sources

To integrate information from different sources and to allow structured formatting as well as textual representation of vulnerability related information, a common format has been derived from the requirements discussed in the previous section as well as from the format provided by the NVD (Byers & Owen, 2019). Figure 10 shows a class diagram of the resulting vulnerability model.

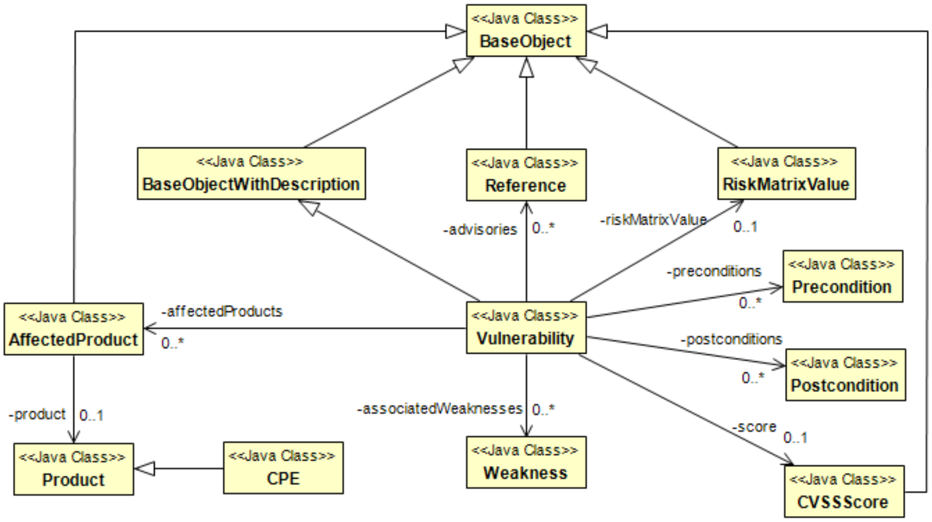


Fig. 10. The Model for Vulnerabilities, Represented as Classes

The central element of this common format is Vulnerability. It contains a description, as well as information on the affected product(s) and the affected version(s). If the vulnerability was retrieved from a CVE entry or other source where an id was assigned, this is represented by by setting an optional ”vulnId” attribute. Moreover, a vulnerability contains information about the associated weakness that it suffers from. The scoring is provided inside the entity CVSSScore and helps in rating vulnerabilities. The entity Reference represents links to advisories and solutions that are correlated with the vulnerability. The risk matrix value is relevant for later risk treatment as it contains exploitability and impact values. Finally, pre- and postcondition yield additional information on prerequisites for an attack as well as actions or conditions that can be achieved after exploiting the system.

In the following this class diagram representation will be explained in more detail by providing a class by class clarification of the entities, relations and attributes.

BaseObject is the main object that contains attributes that are required by all inheriting classes. It is the superclass for the CVSSScore, the Reference class, the Product and the BaseObjectWithDescription. Its attributes are:

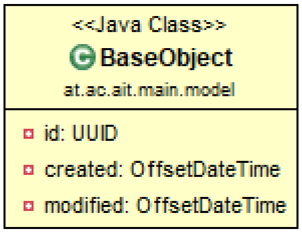


Fig. 11. BaseObject Class

- **id** - to assign a unique identifier for the representation within the database
- **created** - the date when an entry was created
- **modified** - the date when an entry was last modified

These attributes make the inheriting object identifiable and traceable.

BaseObjectWithDescription extends the BaseObject with a description attribute. Currently, the BaseObjectWithDescription class is only inherited by the Vulnerability class and could therefore be omitted. However, as other classes such as Weakness may include a description in the future a decision to keep this class as made.
Due to the fact that it adds a description and extends the BaseObject, it holds the following attributes:

- **id (inherited)** - to assign a unique identifier for the representation within the database
- **created (inherited)** - the date when an entry was created
- **modified (inherited)** - the date when an entry was last modified

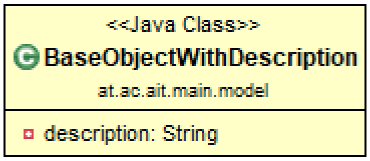


Fig. 12. BaseObjectWithDescription Class

- **description** - the textual description of entities, in this case the description of a vulnerability

Any entity that contains a description in textual form inherits this class. Consequently, the entity Vulnerability utilizes the BaseObjectWithDescription as its superclass, making it possible to include a vulnerability description in its respective entries. Storing and providing this textual information contributes to requirement 2c defined in section 4.3.1.

Please note that the inherited attributes will only be included in the following descriptions where required for understanding, all others will be considered implicit from here on.

Vulnerability describes the central element of the data format. It links with all other major classes. Thus, the association of a vulnerability with a respective weakness, the affected product(s), the CVSSScore, the references to advisories as well as the impact and exploitability values contained in the risk matrix value enable a representation suitable for threat modelling and the subsequent risk management process. The sources for vulnerabilities may vary which is why the CVE class was not directly included into the Vulnerability class, making it possible to incorporate various distinct sources.

As Vulnerability represents the central element, it holds the following attributes:

- **description (inherited)** - the description for the vulnerability presented in online sources, e.g. CVE/NVD, Packetstorm
- **vulnId** - the ID of the vulnerability in the resource (if available)
- **riskMatrixValue** - a wrapper class representing the exploitability and the impact scores for later display on the risk matrix
- **score** - the CVSS Base Score, as it was assigned by the creating entity (only included if available)
- **affectedProducts** - a list of products that are affected by the vulnerability. This information is required to detect vulnerabilities in the respective components
- **advisories** - a list of references to advisories focusing on this vulnerability (only included if available)

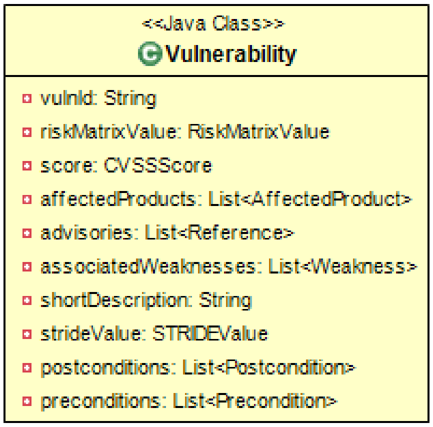

<<Java Class>>
Vulnerability
vulnId: String
riskMatrixValue: RiskMatrixValue
score: CVSSScore
affectedProducts: List<AffectedProduct>
advisories: List<Reference>
associatedWeaknesses: List<Weakness>
shortDescription: String
strideValue: STRIDEValue
postconditions: List<Postcondition>
preconditions: List<Precondition>

Fig. 13. Vulnerability Class

- **associatedWeaknesses** - the weakness(es) specific to this vulnerability. It is a list because some CVE/NVD entries contain multiple CWEs (only included if available)
- **shortDescription** - a short text containing only the vulnerability itself
- **strideValue** - The associated STRIDE value as defined in 3.4.1
- **postconditions** - a list of possible postconditions
- **preconditions** - a list of required preconditions to achieve successful attack

Except for the description of the vulnerability and the affected product including its version(s), all attributes are optional for threat modelling. The goal here is to include all the information that is available and improve the threat model with new contents as long as the minimum requirements listed in section 4.1 are fulfilled.

This class fulfills requirements 1, 2, 3 and 4 as it links all the required as well as the optional information concerning one specific vulnerability in one place. It is able to include the score as well as references. Moreover, the fact that it is optional to add CVEs and represent vulnerabilities in a general form, makes it possible to integrate different data sources.

Weakness contains an associated weakness that a vulnerability suffers from. Its content is derived from the CWE. It only contains the id of a CWE. This is sufficient to reveal the corresponding weakness and provide more detailed information as well as mitigation and detection strategies by allowing security experts to look up the corresponding CWE online utilizing the id. The following attributes are contained within Weakness:

- **cweid** - the id of the weakness in the CWE

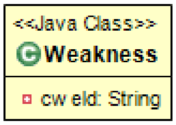


Fig. 14. Weakness Class

The information within a CWE can provide aid when identifying weaknesses for hardware or software systems. Especially when no references to advisories are available, it is possible to retrieve high level mitigation strategies by consulting the CWE dictionary. This class contributes to fulfilling requirement 2b.

AffectedProduct represents a product or range of products that are affected by a vulnerability. It contains the product itself which could be a CPE, as well as a simple string concatenation of vendor and the product name. If there is no range specified then the version of the product itself is considered vulnerable.

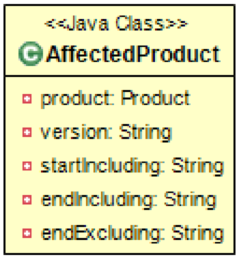


Fig. 15. Affected Product Class

In order to represent the product ranges affected by a vulnerability, the following attributes are necessary:

- **product** - the product that is affected by the vulnerability
- **version** - the vulnerable version of the product
- **startIncluding** - a value indicating the first version that has a certain vulnerability. It does not necessarily have to be a zero-day vulnerability but could rather have been introduced due to updates.
- **endIncluding** - a value indicating the last vulnerable version of a product. If e.g. version 0.17 is vulnerable and there is a fix in version 0.18, endIncluding will be 0.17.
- **endExcluding** - a value indicating the first invulnerable version of a product. If e.g. version 0.17 is vulnerable and there is a fix in version 0.18, then endExcluding will be 0.18.

The AffectedProduct class is relevant, due to the fact that a product could suffer from multiple vulnerabilities but within different versions. Thus, allowing the association of multiple products with multiple vulnerabilities and storing the affected version makes them uniquely identifiable. Therefore, it contributes to fulfilling requirement 2a.

Product presents an instance of a specific product. It contains the title of a product as well as an indicator if it was derived from a CPE entry. This class represents a generic product allowing for attaching arbitrary versions without needing to create one instance per version.

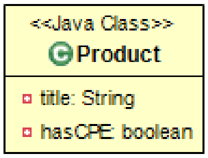


Fig. 16. Product Class

The class holds the following attributes:

- **title** - the title of the product created from vendorname + productname
- **hasCPE** - boolean value indicating whether this product was derived from the CPE

Although the CPE already contains a vast amount of products and platforms, there are products which are not registered with the CPE, or products that were newly developed and consequently not yet included into the CPE. This is why this class shall take care of products that are within the CPE and products that are not. Moreover, this class plays an important role in fulfilling requirement 2a by outlining the product.

CPE, the Common Platform Enumeration has been chosen as a representation for products and platforms as it provides a standardized way of uniquely identifying products. Figure 17 shows the CPE class which is an extension of the Product class. Thus, it is an exact representation of the CPE attributes discussed in section 3.6.2.

This class has been derived from the Common Platform Enumeration: Naming Specification Version 2.3 (Cheikes et al., 2011). Thus, its attributes are:

- **cpeId** - the complete CPE entry as formatted string binding

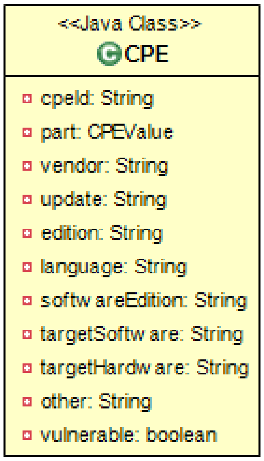


Fig. 17. CPE Class

- **part** - indicates whether a product represents an application, an operating system or a hardware device. It is of type CPEValue which can hold one of three values: *a: application, o: operating system, h: hardware device*
- **vendor** - a value describing the creator of a product
- **update** - a value indicating a specific update, service pack or bug fix release of a given product
- **edition** - a value for backward compatibility with CPE 2.2. It specifies edition-related terms by the product vendor
- **language** - a value describing the language of the user interface of the product, defined in RFC5646
- **softwareEdition** - a value indicating the class of users that the product is aimed at
- **targetSoftware** - a value depicting the environment which the software is tailored to
- **targetHardware** - a value depicting the instruction set architecture. (e.g. x86, x64, Java Virtual Machine, Common Language Runtime, VMs)
- **other** - a value describing any other descriptive information that does not belong to any of the other categories
- **vulnerable** - a value indicating whether there is a known vulnerability for this CPE

Please note that the descriptions for the attributes ranging from **part** to **other** have been derived from Cheikes et al.. This class is used as a representation of known products in a standardized way. It makes the definition of a product more fine granular and contributes to the fulfillment of requirement 2a.

Reference describes advisories on vulnerabilities. In case of a CVE/NVD entry, these references can include additional information about vulnerabilities as well as possible solutions or information on patches. Figure 18 shows the layout of the class.

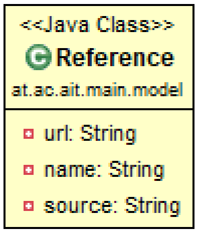


Fig. 18. Reference Class

As it is not possible to cover all the information contained throughout various differently structured webpages, only the hyperlinks to webpages are included to provide further guidance. The information is not further analyzed or processed. Consequently, only the attributes contained in figure 18 are required and will be explained below:

- **url** - the url of the advisory webpage
- **name** - the name of the security advisory which is similar to an id
- **source** - the originator of this information

The information gathered through the references as well as through the associated weaknesses of a vulnerability can complement each other and can give a clearer insight into the threat landscape. This is why this class contributes to requirement 4.

CVSSScore describes a representation of the CVSS Base Score related to vulnerabilities. Some more sophisticated online resources such as the NVD associates vulnerabilities in their repositories with CVSS scores. As mentioned in section 3.6.5, including CVSS Scores where possible can improve the capabilities of a rule and yield information for the risk treatment process. The resulting class is shown in figure 19.

The CVSS Score class has been derived from the format specified by FIRST, which is also described in section 3.6.5. Moreover, the format retrieved from the NVD also played an important role when designing this class. As the attributes were already explained in section 3.6.5 only a short description of the attributes and possible values will be given here:

- **attackVector** - indicates the required location of an attacker. Possible values are: *Network, Adjacent, Local, Physical*

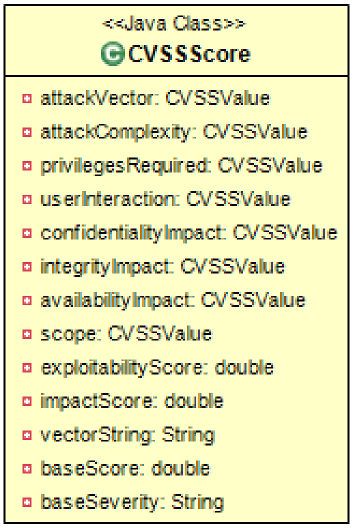

<<Java Class>>
CVSSScore
attackVector: CVSSValue
attackComplexity: CVSSValue
privilegesRequired: CVSSValue
userInteraction: CVSSValue
confidentialityImpact: CVSSValue
integrityImpact: CVSSValue
availabilityImpact: CVSSValue
scope: CVSSValue
exploitabilityScore: double
impactScore: double
vectorString: String
baseScore: double
baseSeverity: String

Fig. 19. CVSS Class

- **attackComplexity** - indicates how much knowledge about a system is required to start an attack. Possible values are: *Low, High*
- **privilegesRequired** - indicates the level of privileges that are required to start an exploit. Possible values are: *None, Low, High*
- **userInteraction** - indicates whether user interaction is required for a successful exploit. Possible values are: *None, Required*
- **confidentialityImpact** - indicates the impact on the confidentiality of information. Possible values are: *None, Low, High*
- **integrityImpact** - indicates the impact on the integrity of information. Possible values are: *None, Low, High*
- **availabilityImpact** - indicates the impact on the availability of a service. Possible values are: *None, Low, High*
- **scope** - indicates whether the scope is (un-)changed. It affects the calculation of the base score. Possible values are: *Unchanged, Changed*
- **exploitabilityScore** - refers to the calculated exploitability score
- **impactScore** - refers to the calculated impact score
- **vectorString** - represents the CVSS Base Score as string vector
- **baseScore** - represents the calculated CVSS Base Score ranging from 0.0 - 10.0
- **baseSeverity** - hints at the severity from 0.0 - 10.0 which is mapped according to the severity rating scheme in table 6. Possible values are: *None, Low, Medium, High, Critical*

The CVSS Score serves as a means to measure qualitative attributes by assigning values to them. These attributes are included in the descriptions above. The respective possible values for CVSSValue are listed below. CVSSScore alongside

CVSSValue enable the inclusion of a scoring system into vulnerabilities and satisfy requirement 3.

CVSSValue as included in the class above represents an enumeration that was derived from the values of the CVSS. It therefore holds all the possible values (FIRST, 2020b).

- NONE
- LOW
- MEDIUM
- HIGH
- NETWORK
- ADJACENT aka. ADJACENT_NETWORK
- LOCAL
- PHYSICAL
- REQUIRED
- UNCHANGED
- CHANGED

The above values represent all the possible settings that a CVSS Base Score contains which affect the exploitability as well as the impact score. Consequently, changing these values results in a different Base Score.

RiskMatrixValue holds the values for exploitability and impact that are also represented in the risk matrix and is relevant for later risk evaluation as well as risk treatment. Figure 20 shows the respective class.

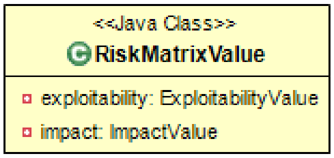


Fig. 20. Risk Matrix Value Class

In order to represent the risk matrix as best as possible the attributes are the following:

- **exploitability** - represents the exploitability value in the risk matrix. The following values can be assigned from the **ExploitabilityValue** enumeration: *Very Low, Low, Medium, High*
- **impact** - represents the impact value in the risk matrix. The following values can be assigned from the **ImpactValue** enumeration: *Negligible, Moderate, Major, Severe*

By assigning exploitability and impact scores, the corresponding field within the risk matrix can be identified, yielding the severity of a vulnerability.

Upon reading the content of entries in Packetstorm and CVE/NVD, two additional properties were identified, namely precondition and postcondition. From this following classes have been derived.

Precondition represents actions or conditions required to exploit a vulnerability. The respective class is shown in figure 21.

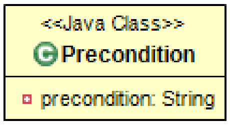


Fig. 21. Precondition Class

The precondition consists of only one attribute which is explained below:

- **precondition** - a precondition specifies an action that must be conducted before an attack can be started. This could be social engineering e.g.persuading a possible victim to visit a specific link or installing malicious files.

This class also utilizes the precondition as id, as each precondition should exist only once.

Postcondition represents possible actions that can be conducted after compromising a system. This class is represented by figure 22.

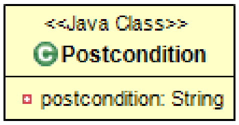


Fig. 22. Postcondition Class

Similar to the class before, there is only one attribute which is described below:

- **postcondition** - a postcondition specifies an action that can be conducted following a successful exploit. This could be arbitrary code execution after exploiting a remote SQL injection vulnerability.

Every postcondition should exist only once and therefore be used as identifier, so that every vulnerability with the same postcondition is linked to the same instance.

Neither do all information sources present the relevant information in the same format nor do they provide the same interfaces. The data requires transformation into the common format which has been presented in this section.

However, the transformation step still requires explanation. Thus, the next section will give insight into the architecture that is used to efficiently transform the information while supporting multiple different sources of information.

4.3.3 Enabling Extendability With the Adapter Pattern

In modern software development an existing software system should not be fundamentally changed but rather extended, e.g. by connecting to new sources of information. Actually, the code on the vendor site cannot be changed anyway. In order to support various information sources to be integrated into one common format, a system needs to be capable of connecting to distinct sources of information while keeping the underlying system untouched. Consequently, the design pattern that was chosen for communicating with different origins of data and for transforming information to a common format is the Adapter Pattern. It enables the integration of different interfaces and data structures and acts as a converter to extend an existing software system. (Freeman et al., 2014)

The Adapter Pattern omits the need of changing existing code by providing interfaces to different sources of information and transferring the communication and transformation logic to a different component. The following examples will give further explanation on how the Adapter Pattern works and why it is necessary for our prototype.

Figure 23 shows an example of a system in which data shall be exchanged. However, their interfaces do not match. Consequently, no data exchange is possible without changing the interface of the system or the vendor system. Usually changing the code in the running system is not possible as there may already be other vendor systems that need to be supported. Changing the interface will work when integrating the new system, but the support for already implemented

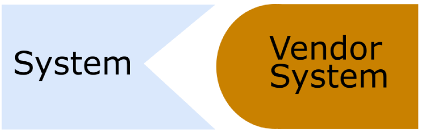


Fig. 23. Example of systems with different interfaces (Freeman et al., 2014)

systems will most likely be lost. This requires an extendable and adaptable approach. An illustration of this can be found in figure 24.

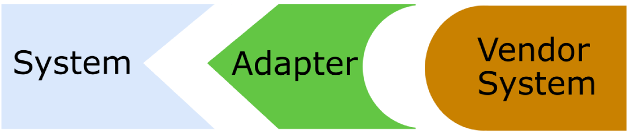


Fig. 24. Example of systems with different interfaces communicating through an adapter (Freeman et al., 2014)

By considering this adapter-based approach there is no need to change the existing system. The adapter can handle the communication between the existing system and the vendor system. It implements an interface that the existing system knows and translates this to a request to the vendor system and vice-versa. A call to the vendor system would look like the following (Freeman et al., 2014):

1. The existing system uses a method provided in the interface of the adapter
2. The adapter translates this request to a request that fits the specification of the vendor system
3. The vendor system returns the desired data to the adapter
4. The adapter transforms the retrieved information into a format that can be understood by the existing system and returns the converted information
5. The existing system utilizes this information and processes it

The adapter works as a middleman taking over communication. The client receives the results without knowing the exact class that implements the adapter and handles the translation of the exchanged information (Freeman et al., 2014). The client does not even need to know that there is communication with an external entity. Figure 25 shows how adapters work as connector to support multiple different vendor systems.

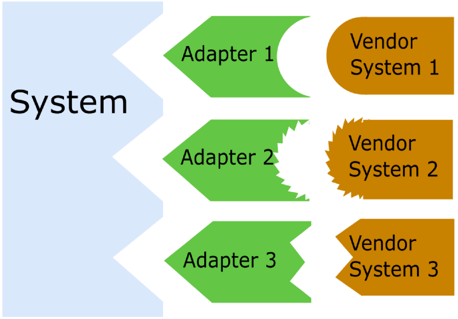


Fig. 25. Example of a system utilizing multiple adapters based on Freeman et al.

As can be seen in figure 25, the interface between the adapters and the existing system is always the same. The system only requires a call to the interface which is then handled by the adapters. The adapters extend the provided interface which allows them to implement connectors to different vendor systems. Thus, the original system is kept untouched while providing extendability. (Freeman et al., 2014)

Requirement 1 aims at the integration of various information sources and thus requires an extendable solution in order to include different interfaces. The Adapter Pattern can add this functionality and take over the communication and thus contribute to fulfilling this requirement. Figure 26 illustrates the process with regard to this master thesis.

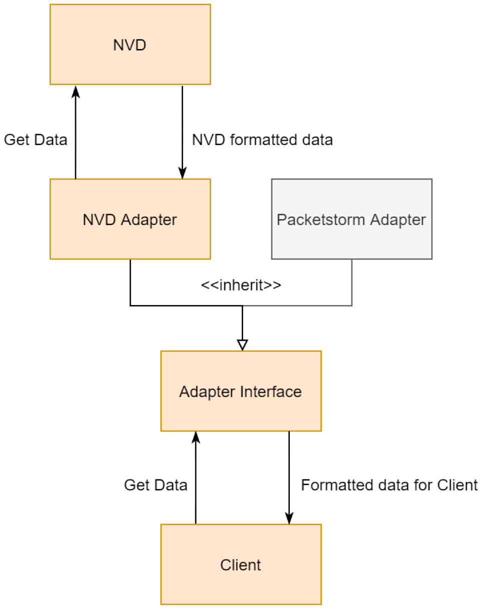


Fig. 26. Illustration of the Adapter Pattern for Connecting to Vulnerability Databases based on (Freeman et al., 2014)

The image shows a client requesting data from the NVD by setting off a call to the adapter interface. The adapter interface which is inherited by the NVD Adapter then calls the NVD database through the specified interface and retrieves the data in the NVD format. For the client to utilize this information, the adapter takes care of transforming the NVD format to the model that was specified in section 4.3.2. This way, no extra code within the application is necessary. The data is then ready for further processing. The process for retrieving data from Packetstorm is analogous to the one presented here.

The Adapter Pattern allows for a simple extendability in the future when additional databases will be added. Some of these databases contain additional information while others focus on the essentials and the minimum requirements when identifying vulnerabilities. One of these additions is the CVSS Score which also needs to undergo a transformation in order to fit into the custom vulnerability model. The logic behind this transformation will be explained in the following section.

4.3.4 Converting CVSS to Risk Matrix

Section 3.3.3 introduced the adaptable risk matrix meant for company specific risk management. When analyzing the process of generating rules, one needs to consider that every rule requires an impact and an exploitability value. These attributes can be found in the CVSS, whose exploitability shall be mapped to the exploitability in the risk matrix columns while the impact inside the CVSS can represent the impact rows in the risk matrix. Therefore, the results of the calculated impact and exploitability scores can be utilized.

In order to find an accurate mapping, the severity ratings of a CVSS require examination. To provide a better overview, severity ratings are repeated below in table 9.

Rating	**CVSS Score**
None	0.0
Low	0.1 - 3.9
Medium	4.0 - 6.9
High	7.0 - 8.9
Critical	9.0 - 10.0

Table 9. Severity Rating (FIRST, 2020b)

As mentioned before in section 3.3.3, the risk matrix contains values between 1 and 5. Table 9 also holds 5 possible values: *None, Low, Medium, High, Critical*. Therefrom, it is feasible to derive a mapping that satisfies the allowed values between 1 and 5 for the risk matrix. A possible mapping is described in table 10.

However, simply mapping severity is not sufficient. This section has already dealt with representing the impact and exploitability scores within the risk matrix. Accordingly, the maximum values for impact and exploitability need to be determined. By applying the formula for calculating the impact and exploitability scores as well as setting the attributes to the worst case values specified in table 5 the maximum impact and exploitability scores can be derived.

CVSS Score	Risk Matrix Severity
0.0	1
0.1 - 3.9	2
4.0 - 6.9	3
7.0 - 8.9	4
9.0 - 10.0	5

Table 10. Severity Score to Risk Matrix Severity Rating

In order to receive the worst case result for exploitability, the following values are required:

- **Attack Vector** - *Network* - 0.85
- **Attack Complexity** - *Low* - 0.77
- **Privileges Required** - *None* - 0.85
- **User Interaction** - *None* - 0.85

Applying formula 1 for calculating the exploitability score from section 3.6.5 and setting the values from above, leads to the following result:

$$\textbf{Exploitability} = 8.22 * 0.85 * 0.77 * 0.85 * 0.85 \approx 3.887 \approx \mathbf{3.9} \tag{13}$$

The resulting exploitability score represents the maximum possible value for this metric. It can be divided into four categories that resemble the structure of the risk matrix. These values range from 0.0 to 3.9. Consequently, the score can be mapped to the values associated with the exploitability inside the risk matrix. Table 11 shows the exploitability mapping.

Exploitability Score	Risk Matrix Exploitability
0.0 - 0.65	Very Low
0.66 - 1.95	Low
1.96 - 3.25	Medium
3.26 - 3.9	High

Table 11. Exploitability Score to Risk Matrix Exploitability Mapping

Due to the distribution of the CVSS score criticality as depicted in table 10, it was decided to divide 3.9 by 3 to retrieve the sizes of the appropriate ranges for the mapping. As a consequence, a scope of 3 ranges - each of a size of 1.3 - were derived. Two of the resulting values were then utilized as the range for **Low** and **Medium**. The third range was split into two values of 0.65 ($2 * 0.65 = 1.3$) where **Very Low** and **High** represent the lower and upper bounds for the values.

The same procedure can be conducted with the impact score. The maximum for its attributes are listed below.

- **Confidentiality** - *High* - 0.56
- **Integrity** - *High* - 0.56
- **Availability** - *High* - 0.56

By applying formula 3 for calculating the impact score from section 3.6.5 and setting the values from above, the following result is received for the impact subscore (ISS):

$$\mathbf{ISS} = 1 - [(1 - 0.56) * (1 - 0.56) * (1 - 0.56)] = \mathbf{0.914816} \tag{14}$$

To calculate the maximum result, we need to assume a change of scope. Therefore, formula 5a applies:

$$\mathbf{Impact} = 7.52*(0.914816-0.029)-3.25*(0.914816-0.02)^{15} \approx 6.0477 \approx \mathbf{6} \tag{15}$$

Again the maximum value can be divided into four categories resembling the impact rows inside the risk matrix. The values range from 0.0 to 6.0. The mapping was conducted in the same way as for the exploitability. Table 12 displays the impact mappings.

Impact Score	**Risk Matrix Impact**
0.0 - 1.0	Negligible
1.1 - 3.0	Moderate
3.1 - 5.0	Major
5.1 - 6.0	Severe

Table 12. Impact Score to Risk Matrix Impact Mapping

These resulting exploitability and impact values can afterwards be used to define the severity within the risk matrix. Risk is a subjective matter, which is why distinct use cases may require different risk evaluation and risk treatment strategies. Different individuals also consider risk differently (ISO/TC 262 Risk management, 2018). Of course, a CVSS Score could be mapped to the generic risk matrix described in section 3.3.3 but a more accurate mapping can be obtained by adapting the existing risk matrix according to the formulas used within the CVSS.

The step size was the main factor that influenced the resulting matrix. As lower and upper bounds were only half in size and because it was necessary to represent the minimum value of 0.0 and the maximum value of 10.0, these minimum and maximum values were utilized. In terms of exploitability this means 0.0 and 3.9, whereas for the impact 0.0 and 6.0 were used. As the ranges for values inside are specified in both directions, the mean was defined to be the basis for calculation. This means that for exploitability **Low** is represented by 1.3 and **Medium** by 2.6. As far as the impact is concerned, **Moderate** is attributed a value of 2 and **Major** is defined as 4.

Now that the mapping for the exploitability and impact values has been explained, the results of the exploitability and impact score need to be combined in order to retrieve the final risk matrix. As the worst case is considered here, the CVSS Base Score formula that includes a scope change must be included in the calculation. Consequently, the values for the risk matrix are calculated according to formula 8a.

When considering an exploitability of *Very Low* and an impact of *Severe*, the result is the following:

$$\textbf{Base Score} = Roundup(Minimum[1.08 * (6.0 + 0.0), 10]) = \mathbf{6.5} \tag{16}$$

This can be done for every element in the risk matrix. Another example for calculating the Base Score for the matrix is shown below with an exploitability of *Low* and an impact of *Severe*:

$$\textbf{Base Score} = Roundup(Minimum[1.08 * (6.0 + 1.3), 10]) = \mathbf{7.9} \tag{17}$$

This calculation can be conducted by computing the Base Scores with all respective impact and exploitability values. The result is a 4x4 risk matrix presented in table 13.

		Exploitability			
		Very Low	Low	Medium	High
Impact	Severe	6.5	7.9	9.3	10.0
	Major	4.4	5.8	7.2	8.6
	Moderate	2.2	3.6	5.0	6.4
	Negligible	0.0	1.5	2.9	4.3

Table 13. CVSS based Risk Matrix

Due to the definition of the CVSS score as well as its calculation, impact has more effect on the overall score than exploitability, which is made evident in the corresponding table cells. In the CVSS, high exploitability but negligible impact receives a lower score than severe impact and very low exploitability. Please note that the risk matrix specified in table 13 contains crisp borders which may lead to a severity classification that is slightly higher or lower than the original CVSS score.

Simply representing the CVSS score mapped inside a risk table is still not sufficient. To represent the CVSS based risk matrix within the range of 1 to 5, one last conversion is required. This conversion can be deduced from table 10. By mapping the values from the CVSS score to severity, a conformant risk matrix can be derived. The resulting matrix is presented in table 14.

		Exploitability			
		Very Low	Low	Medium	High
Impact	Severe	3	4	5	5
	Major	3	3	4	4
	Moderate	2	2	3	3
	Negligible	1	2	2	3

Table 14. CVSS Risk Matrix mapped to values from 1 to 5

Table 14 shows results that differ from the generic risk matrix presented in section 3.3. In contrast to the generic matrix, the mapped CVSS-based risk matrix is asymmetric due to the fact that the impact takes more effect. This is also reflected by table 11 and table 12. While the steps have a size of 0.65 and 1.3 for the exploitability, the steps of 1.0 and 2.0 for the impact are bigger. Generally, the newly created matrix treats risk more critically. However, the score must be derived depending on an organization's needs. Therefore, this matrix should be considered a suggestion. Different organizations may configure their risk matrices differently.

Although the unmapped matrix in table 13 contains a 0.0 entry, this value is mapped to 1. This is not only for reasons of conforming to the constraint of allowing only values between 1 to 5 inside the risk matrix, but also to observe the risk. A value of 0.0, which is defined as *none* in CVSS, would mean that there is no risk although there is a known vulnerability. Consequently, a mapped value of 1 defines that the given vulnerability should still be inspected and be considered in the risk treatment process.

As the risk treatment process not only depends on quantifying risks correlated with vulnerabilities or threats, the type of threat posed to the system is also highly relevant. This is why a rule also requires a categorization which the next section will elaborate on.

4.3.5 CVSS Based CIA to STRIDE

When taking a closer look at the impact score one may notice that it is constituted of 3 attributes: **C**onfidentiality, **I**ntegrity and **A**vailability (CIA) represent the resulting impact score. But the threat modelling approach that is utilized within this master thesis is based on STRIDE.

On the one hand, STRIDE is seen as an extension of the CIA model (Lautenbach & Islam, 2016) and allows some attributes to be directly mapped. In contrast to STRIDE which portrays threats from an attackers perspective, the CIA model considers security attributes. These attributes are illustrated in figure 27.

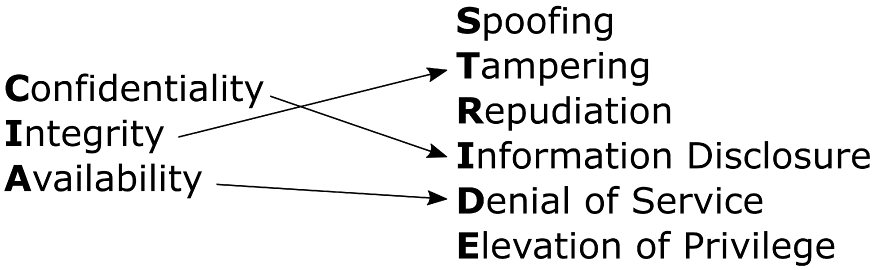


Fig. 27. CIA to STRIDE mapping (Hamad, 2020)

On the other hand, STRIDE contains some additional attributes that cannot be simply inferred. These attributes are spoofing, repudiation and elevation of privilege.

Although there are extensions to the CIA model such as the CIA3 (Hamad, 2020) which cover more security attributes and enable mapping between STRIDE and CIA3, these extensions are not available within the CVSS score as it only supports CIA.

The fact that not all attributes from STRIDE can be represented within the CIA model, makes the mapping more complicated and would require additional intelligence when assigning a category to a vulnerability. A possible solution for this issue would be to search the vulnerability description for terms like *"spoofing"*, *"tampering"*, or *"elevation of privilege"*. Although this might produce some results, different formulations may affect the search results negatively. Furthermore, the presence of the term *"spoofing"* does not necessarily describe a spoofing vulnerability but could also describe a postcondition resulting in an incorrect classification of the vulnerability.

Another approach to solve this mapping issues could be to train an algorithm into classifying vulnerability descriptions into STRIDE categories. Therefore, a machine learning approach would be required alongside supervised training.

As the categorization of threats and vulnerabilities is not the main topic of this master thesis, the machine learning approach to cover this issue is considered out of scope but may be adopted in future research.

Consequently, a different solution was selected to deal with this problematique. As the CVSS includes the values *none, low, high* for confidentiality, integrity and availability, a mapping of these three categories can be achieved. By selecting the highest rated category from the CIA model, information disclosure, tampering, and denial-of-service categories can be assigned to the resulting rule.

Should a vulnerability have multiple maximum values, then there must be a user controlled assignment to a rule. However, this approach ignores spoofing, repudiation and elevation of privilege. Consequently a manual review might be necessary.

Now that all the relevant pieces for the model have been discussed, these need to be brought together. The following section provides an explanation on how these components work together and how manual rule creation can be improved by automation.

4.3.6 An Automated Rule Creation Process

The rule creation process starts with a basic step of **data collection**. An HTTP client is utilized to connect to existing vulnerability or exploit databases and retrieve data from them. In case of the CVE entries within the NVD this data is exposed via a REST API. It can be accessed through parameterized URLs and returns NVD entries in JSON format based on the specified query params.

In case of Packetstorm, the data is not presented through a REST API, but needs to be accessed with a webcrawler. Therefore, the HTML needs to be analyzed and the entries are extracted programmatically in plain text format.

In order to keep the required traffic to a minimum and for testing purposes, the initial requests are stored on disk for further processing.

Storing the data in a file before undergoing further analysis is mainly because of two reasons. Firstly, for simplifying the implementation process. Writing a programme conducting data extraction, may yield coding errors. This is either due to a misconception of an attribute which may be mapped incorrectly, or due to libraries that throw an exception in a case that the programmer did not consider. Moreover, establishing a connection, downloading all the required information and afterwards running into an error is more time consuming than working with a local file.
Secondly, a part of the data contained within these files can be utilized as train-

ing and test sets for the classifiers used in Named Entity Recognition.

Of course, this step of storing information will be omitted in the future and direct connections with the databases will be established without taking further steps in between. But for the course of this master thesis keeping this information stored is considered as required.

Once the information has been stored inside a file, the classifiers for the Named Entity Recognition need to be trained. This is not a common step in the rule creation process but is rather done once. Consequently, only the resulting entity extractor will be utilized in the extraction step. The training of the classifiers will be covered in section 5.

The **entity extraction** step requires the data to be loaded from disk. This can be done using the adapter pattern to allow for different formats. Depending on the information source and its quality regarding the structure, it can either be directly mapped to the vulnerability representation model or be staged for further analysis.

When further analysis is required, the Named Entity Recognition step is executed. This is done to analyze unstructured text and retrieve categorized portions of the text - named entities. In order to keep the database clean and to only provide valid and consistent data, extracted information requires validation.

As information is extracted automatically when utilizing a machine learning approach, there will likely be errors. To counter this, a validation step is introduced. This way the error rate can be reduced, thus keeping the quality of the data inside the database high. Moreover, validation can provide an insight into aspects that may have been neglected or unnoticed during training. Therefore, the information gathered during this step will flow into improving the classifiers.

In the validation step, a human entity reviews the extracted data and modifies the results where necessary before they are mapped to the vulnerability model. Once the transformation is complete, the vulnerability can be inserted into the database where it is linked with existing products and weaknesses. A flow diagram outlining the described topics is presented in figure 28.

Once entities have been extracted they will be stored inside a relational database constructed of the entities specified in the vulnerability model. This enables a static representation of the information in a common format capable of representing various information sources. From this static format it is feasible to generate rules without the need of manual user interaction. For demonstration purposes, the GUI of the prototype in section 5.1.5 will display the rules before storing the data into the database. However, in the future the static representation based on the vulnerability model will enable the automated creation of

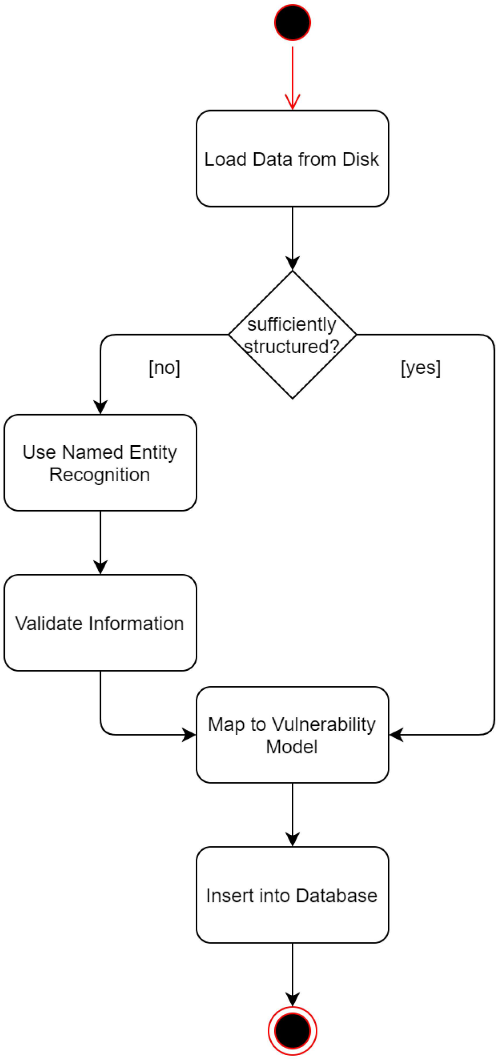


Fig. 28. Entity Extraction

rules even for a different or adapted grammar.

Therefore, the **rule generation** step can be kept simple. The vulnerability data is loaded from the database in the common vulnerability format. Afterwards, the generator component, which will be explained in section 5, statically generates rules according to the provided data.

Once the three steps of **data collection**, **entity extraction** and **rule generation** have been conducted, these generated rules can be integrated into the threat model leveraging its capabilities as a digital twin of real-world systems.

The following section elaborates on the methodological approach and what influenced the decision for technologies for the vulnerability model and the rule creation process.

4.4 Methodological Approach

In order for the reader to comprehend what factors influenced the decisions inside this thesis, the following sections shall provide insight. The requirements for creating a rule had to be specified before deciding on the platforms to use. The focus was put on platforms that offer data which is valid for longer time periods. After this, connections to the platforms were established to find out in what form the information is returned to decide on the mechanisms to use. From this a common format was derived. However, not all information was directly usable. Therefore, some initial data was downloaded and annotated to train a named entity extractor. Finally, possible formulations of a rule from the provided data were evaluated.

In the following, the methodological approach to building a rule generator from online resources will be explained.

4.4.1 Identifying Requirements for Rule Creation

Before starting off with online research on the information that can be integrated, the minimum requirements for uniquely identifying a vulnerability within a component had to be specified. Therefore, a feasibility study was conducted and the entries in the CVE were analyzed. Originally only three attributes, namely **product**, **version** and **vulnerability** were considered as relevant. These three attributes are contained within every CVE entry.

Afterwards the capabilities of the existing rule structure were analyzed in order to decide whether a representation of component-based vulnerabilities is possible. As the grammar allows for specifying arbitrary attributes, it was found feasible to create rules according to CVE entries.

Once the task was considered feasible, multiple different platforms were analyzed and their offerings were compared to the capabilities of threat modelling.

4.4.2 Analyzing Platform Offerings

In this phase existing blogs, threat intelligence platforms, and vulnerability databases were consulted. The aim was to learn about their capabilities and to find out about the information they contain.

When considering component-based security targeted at specific products, blogs and articles did not yield sufficient results. They were mostly about security in general an the abstract ideas of what a system should (not) look like. Articles represented a more abstract and high level form of threat intelligence. Moreover, they contained information on attacks that were attempted, exploits that happened or attacker methodologies, mostly without component specific data.

Wang and Chow provide a list of articles and blogs. These served as a starting point for the search. Among the consulted resources were: securelist[7], fireeye[8], krebsonsecurity[9] and schneier[10]. Although this information is suited for high level decision making, it does not fulfill the requirements for representing specific components and their vulnerabilities and has thus not been included in the further process of generating rules.

The next type of resource that was analyzed were threat intelligence platforms such as AlienVault OTX[11] and MISP[12]. These present vital information for running systems in the form of indicators of compromise. The existence of these indicators of compromise may hint at a possible intrusion or existence of malware in a system, e.g. an IP address, a file hash, or a specific file name. These platforms and their data are necessary for running systems, they are not applicable to the current state of threat modelling, as it is based on the system architecture as well as the utilized components.

Short-lived threats such as malicious IP addresses, will not be included in the threat model, as these are often valid only for short time periods. Furthermore, also long-lived information such as file hashes and file names that are used in anti-virus software will not be included as well, for they do not affect the system architecture. Of course, firewall configurations and anti-virus software need to be considered during system development. Therefore this information should only affect the system design on a high level. It should be included but only for the conceptualisation of a system as it is too specific and probably already invalid after a short time period.

[7] `https://securelist.com/`

[8] `https://www.fireeye.com/`

[9] `https://krebsonsecurity.com/`

[10] `https://www.schneier.com/`

[11] `https://otx.alienvault.com/`

[12] `https://www.misp-project.org/`

The last area for searching for information was vulnerability intelligence. Manifold, databases representing vulnerabilities and exploits are in use. The starting point in this area was the CVE, which provided the structure of **product**, **version** as well as **vulnerability** and led to the NVD. The latter provided a much higher amount of additional information including severity ratings for vulnerabilities. This facilitated the use of more information within a rule. Furthermore, as the CPE and the associated weakness are included within an entry inside the NVD in a structured format, the first idea was to integrate this information without the need for natural language processing.

Unfortunately, the CVE/NVD does not cover all vulnerabilities but only the ones that have CVE IDs assigned by CNAs. This is why more research in this area was conducted. Among these resources were Packetstorm[13], 0day[14], VulnDB[15], VulDB[16] and Rapid7[17]. As VulnDB, VulDB and Rapid7 represent vulnerability and exploit databases that require payment, these have currently not been added to the rule generation process. Consequently, Packetstorm and 0day have been selected for further analysis.

Although 0day claims to be *"the ultimate database of exploits and vulnerabilities and a great resource for vulnerability researchers and security professionals"*(0day Today Team, 2020) they also offer exploit tools that require payment. Therefore, this resource was omitted for ethical reasons. This is why Packetstorm was chosen as second resource.

Packetstorm entries contain the **product**, **version** and **vulnerability** in textual form and require further analysis with natural language processing. Unfortunately, Packetstorm does not assign CPEs or severity scores. But still the minimum requirements for representing rules are fulfilled. When analyzing the entries within Packetstorm, it was discovered that pre- and postconditions could also be extracted from some of the entries. As many CVE entries also reflect both of the prior within their description this idea was adopted to be also integrated in the analysis of CVE entries. This is why CVE/NVD entries undergo the entity extraction step as well, although they were originally not considered relevant for text analysis.

But before going into detail about the extraction, the data had to be collected.

4.4.3 Connecting to Platforms

As the NVD offered their data feeds within JSON format, the original idea was to download and integrate these JSON files by extracting the contained information. Although the goal of retrieving vulnerability data from these JSON files

[13] `https://packetstormsecurity.com/`

[14] `https://en.0day.today/`

[15] `https://vulndb.cyberriskanalytics.com/`

[16] `https://vuldb.com/de/`

[17] `https://www.rapid7.com/`

could be fulfilled, a better solution was found soon. The NVD offers access to a REST API through which all CVE entries can be queried. This makes it easier to retrieve specific entries or entries according to their modification dates, as CVE entries may also be modified after being published.

Connecting to the REST API is simple and clean. But there are some restrictions. The NVD tries to prohibit denial-of-service attacks and thus, limits the amount of requests that can be executed. Consequently, there should be a few seconds of idle time within the application before setting off a new request.(Byers & Owen, 2019)

As Packetstorm does not provide a REST API or something similar, the connection to Packetstorm was established with a simple HTTP client and a webcrawler. By studying the structure of the HTML, the important tags alongside their identifiers could be found and data extracted therefrom.

To join this information together, a common format had to be defined.

4.4.4 Common Format

When identifying the minimum requirements for representing component-based vulnerabilities within rules, there was already a rough idea of a format. But the analysis of the format returned from the NVD (Byers & Owen, 2019) gave surprising insights into a more sophisticated representation of vulnerability data. Instead of simply mapping a product alongside its version as two attributes, it provided additional value in including the Common Platform Enumeration, as well as ranges of version of an affected product. Furthermore, the presence of possible solutions and identified weaknesses affected the final format considerably.

The inclusion of a severity rating was not regarded feasible in the beginning as only text analysis of descriptions was considered which did not include any information on the severity of events. Originally the value for impact and exploitability of a rule or in this case of a vulnerability should be set manually. But the CVSS score provided in NVD entries allowed for the automatic derival of impact and exploitability values for risk treatment. It took some effort to find a mapping meeting the conditions of the risk matrix. There were basically three ideas:

1. The first one was assigning custom values to each of the attributes used in the metrics of a CVSS score. The sum of either the exploitability or the impact values should then be mapped to the values utilized within the risk matrix depending on a certain threshold. This idea was discarded once the CVSS specification (FIRST, 2020b) was consulted, which contained the real values for each of the attributes. As the mapping would not have been correct, it was not considered scientific.

2. As the maximum values are 3.9 for the exploitability and 6.0 for the impact score, a mapping from 1 to 4 was considered to match the requirements of the risk matrix. The idea behind this was to utilize percentual values in order to map 6.0 to a maximum of 4.0 by simply dividing 4.0 by 6.0, and analogously dividing 4.0 by 3.9. This enabled the mapping of arbitrary values within the allowed range to the four impact and exploitability values contained in the risk matrix. But defining fitting thresholds for these transformed values seemed to to be a trial and error approach which is why this idea was discarded as well.
3. In order to not over-complicate the mapping between the CVSS score and the risk matrix, the score was simply divided by the number of possible impact and exploitability values. This way an intermediate risk matrix could be derived. By assigning the original severity ratings from *none* to *critical* to values from 1 to 5 a mapping corresponding to the CVSS could be derived, thus reflecting a CVSS correctly within the risk matrix.

Once a common format had been defined, the next step was to conduct some experiments concerning data extraction from unstructured text.

4.4.5 Data Extraction

Before conducting data extraction, it was necessary to define labels required for later categorization of the named entities. Originally the following values were considered relevant:

- Vendor
- Software
- Hardware
- Version
- Vulnerability
- Postcondition

A small training set containing approximately 40 sentences was manually annotated with these labels. This resulting training set was then used to train a CRF classifier in the Stanford NER tool. Afterwards the classifiers were tested against arbitrary entries of Packetstorm followed by a qualitative analysis. Consequently, the resulting categories for the named entities were evaluated. More details about CRF classifiers can be found in (Finkel et al., 2005).

The results for version and vulnerability were promising. As far as the postcondition is concerned, there was not enough training data containing this label to draw any assumptions. However, this configuration turned out to be problematic as it requires the classifiers to distinguish between vendors and software/hardware. As it often manual research is necessary to find out the vendor of a product as well as differentiating between a software and hardware product, these three values were omitted and combined into one single entity, namely the product. Moreover, during the qualitative analysis it was found that some entries contain preconditions, which proved to be interesting to include for the future. Consequently, the existing labels were adapted to the following:

- Product
- Version
- Vulnerability
- Precondition
- Postcondition

The original 40 sentences were then re-annotated with the new labels where all labels containing vendor, software or hardware were renamed to product. The resulting model yielded better results on Packetstorm entries than the original model. In order to evaluate the effectiveness on CVE/NVD entries the trained classifiers were also applied to CVE/NVD descriptions. While the classifiers performed well on text that was formulated in a similar way as the text contained within Packetstorm entries, it performed poorly on entries that deviated from that structure.

Due to the fact that CVE/NVD entries contain structured information the product, version and vulnerabilities do not lie in the focus of the extraction for these entries. Consequently, these entries require more focus on preconditions and postconditions.

Therefore, we made a decision to train three distinct entity extractors. The first one focuses on Packetstorm entries only. The second one is trained only on CVE/NVD entries where focus was put on entries containing pre- and postconditions. The third and final one contains a combination of both. The results of these three distinct models will be discussed and evaluated in section 5.2.

As the final result of the envisaged solution requires the generation of rules suitable for threat analysis, the following section will discuss potential approaches to achieve this goal.

4.4.6 A Rule Structure Supporting Component-Based Threat Modelling

The main topic for generating rules was to find a way to represent the affected product and version as complete and compact as possible within a rule. As the entries within the selected vulnerability databases are specific to components rather than connections between components, it has turned out that these rules can be formulated in a very simple way. To identify the vulnerability for a component the ”type” and the ”tv” attributes containing product and version information must be contained within a rule. The ”connector” related information can be omitted. In order to find a suitable representation for the provided vulnerabilities in the form of rules, three different approaches have been identified and analyzed:

1. Formulate the product and the version as the type of the component
2. Formulate the product as the type of the component and the version as a tagged value

3. Formulate the product as well as the version as tagged values of any type

Taking a closer look at approach number 1 yields a number of problems as the approach is not dynamic enough. An example for such a rule is displayed below:

```
Type(”BIG-IP 15.0.0”)
```

Although the rule above is capable of revealing threats within BIG-IP 15.0.0, it cannot cover version ranges, as it is specific to only version 15.0.0. Consequently, one would need to formulate one rule for every affected product version, which will result in a vast amount of rules. Moreover, it would be necessary to create diagram elements for every version that is available.

Approach number 2 yields an improvement over the previously explained type of rule. An example is given below:

```
Type(”BIG-IP”).tv(version = ”15.0.0”)
```

By removing the version from the ”type” attribute it is possible to omit the necessity of creating one rule per element as well as the creation of new diagram elements for every version. This approach only requires one new element and one new rule to be defined.

But we can still do better by utilizing approach number 3, exemplified in the following illustration:

```
Type(”ANY”).tv(product = ”BIG-IP” AND version = ”15.0.0”)
```

In this example, the product name has been moved from ”type” to the tagged value ”product” which eliminates the need of creating specially crafted elements.

As the third approach produced the least overhead, it was selected for later use in the prototype.

This section discussed a model for storing vulnerabilities, methods to convert various formats into a fitting representation and the methodological approach of the single steps of this thesis. These topics serve as a basis for the prototype discussed in the following section.

5 Prototype Implementation

In the previous sections we discussed the basis and the technology that the prototype builds upon. Furthermore, a model for representation of vulnerabilities and mappings of attributes were elaborated on. Now that the foundations of the prototype have been explained, this section will cover the architecture behind the prototype, explaining how to establish connections to the online repositories and will give details about the annotation process as a foundation for Named Entity Recognition. Moreover, new keywords simplifying rule creation will be introduced. This will be followed by a description of the prototype's GUI. Finally, resulting rules and measurements will be discussed.

5.1 Architecture

All the knowledge that was gained during literature review, the analysis of the online resources, and the model and methods described in section 4 have been implemented inside a prototype. An illustration of the setup and inter-dependencies is given in figure 29.

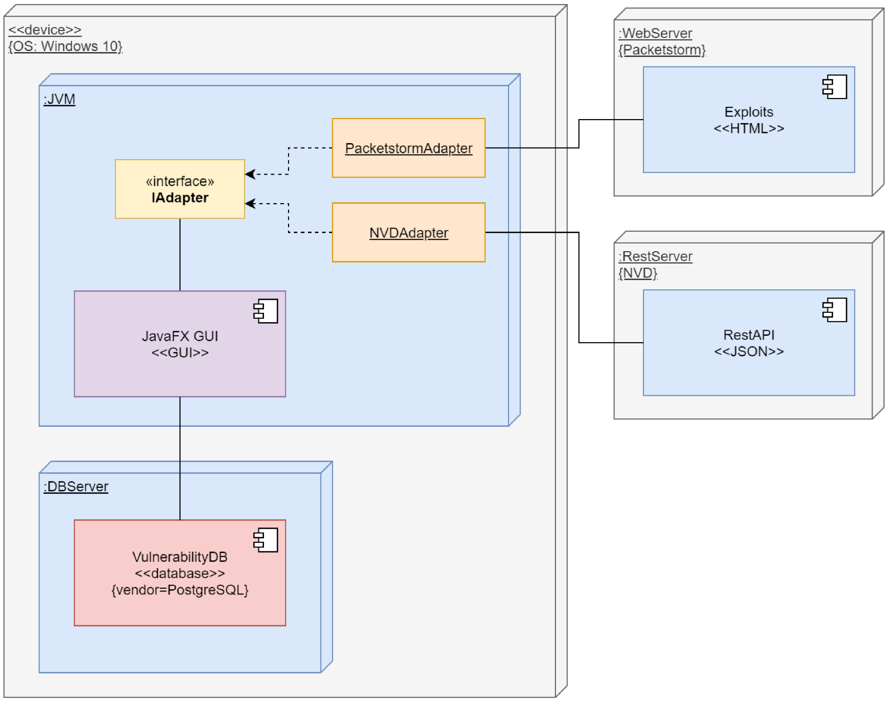


Fig. 29. System architecture overview of the prototype

The whole development of the prototype was conducted on a Windows 10 machine. However, the implementation was written in Java 8 and is, therefore, platform independent. Another reason behind the utilization of Java is the fact that most of the development in this area at AIT has been conducted within Java. Consequently, a possible future integration will be rather seamless. Gradle[18] is used as a build tool for the prototype allowing for a simple setup anywhere.

In order to give the user a clearly arranged layout, a GUI has been created with JavaFX[19], enabling a painless modification of rules and their input data. JavaFX is a library that allows for the creation of a GUI represented as XML file. With a tool called Scenebuilder[20] it is possible to drag and drop GUI components into an application window that can later be utilized within an application. JavaFX allows for a fast GUI definition. Although the diagram contains only a JavaFX GUI component, it actually also consists of a GUI controller which fills the GUI with information. Moreover, it transforms the user validated information into the vulnerability model suitable for storage.

Depending on the settings, the GUI controller holds the interface IAdapter which is of type PacketstormAdapter or NVDAdapter. The approach described here is basically the same as the one described in section 4.3.3 with the only difference that in our case the "Client" is the GUI controller. Both adapter implementations work in the same way. They provide a method to connect to the database and store the retrieved resources inside a file.

Another method allows for loading this stored information and then transform it into a representation that the GUI can handle. This step also involves Named Entity Recognition. As literature research has revealed that many approaches using information extraction (Vadapalli et al., 2018; Joshi et al., 2013; Satyapanich et al., 2019; Syed et al., 2016) depend on Stanford CoreNLP (Manning et al., 2014), it has been decided to also go down that road. There are also other NLP libraries such as NLTK (Bird et al., 2009) and SpaCy (Honnibal & Montani, 2017), which are commonly used. However, it has to be taken into account that they rely on python as a basis and have therefore not been considered for the implementation.

Both adapter implementations contain a method to store the extracted information that is displayed in the GUI. A PostgreSQL database is used for persistent storage. It is an open source database and is already utilized by AIT's implementation of threat modelling that we build upon here. As the prototype works with objects, Hibernate[21] is used for Object Relational Mapping (ORM). ORM is an *"attempt to make incompatible systems cooperate, communicate, and exchange*

[18] `https://docs.gradle.org/current/userguide/userguide.html`

[19] `https://openjfx.io/`

[20] `https://www.oracle.com/java/technologies/javafxscenebuilder-1x-archive-downloads.html`

[21] `https://hibernate.org/orm/`

information" (Kouraklis, 2019, p. 1). More specifically, it enables mapping the Java objects to a relational database and vice-versa.

The process behind the prototype is a simple **ETL** (extract, transform, load) process of the data warehouse (Mehler-Bicher et al., 2019). The **extraction** step is conducted in the phase where the online repositories are queried for their information in its original format. The **transformation** consists of two sub-steps. For the training and its respective annotation process, data cleansing was initially performed on the dataset in order to only allow reasonable documents. Single outliers could drastically reduce the measured performance of the classifier if contained in the test set and would never be useful if inside the training set. The other sub-step concerning the **transformation** is the application of the NER classifier on the documents. The results are then put into the format suitable for vulnerability representation. Finally, the prototype allows for **loading** data that has been validated by an expert into the database by clicking a "save" button.

This section dealt with a description of the architecture, technology choices and concepts forming the foundation of the prototype. The following sections will elaborate on the prototype implementation in more detail.

5.1.1 Connecting to NVD and Packetstorm

As the application requires data from online resources, connections have to be established. This is realized utilizing the adapter pattern. When connecting to the NVD or Packetstorm different instantiations of the adapter are used while still communicating through the same interface with the core application. This is due to the fact that the data for NVD and Packetstorm need to be accessed in different ways.

5.1.1.1 Connecting to NVD

As far as the NVD is concerned, REST services can be used to retrieve vulnerabilities. This way all the resources (vulnerabilities) can be accessed by setting parameters within a URL, yielding a result in JSON format. In order to retrieve a specific CVE entry one can conduct a query in the following form (Byers & Owen, 2019):

https://services.nvd.nist.gov/rest/json/cve/1.0/<**cveId**>

<**cveId**> denotes a placeholder for an existing CVE ID. However, retrieving specific CVE entries is not sufficient as the end-user will most likely not know about the CVE ID regarding a certain product. Moreover, we need to know the vulnerable products in order to identify their vulnerabilities. Therefore, a collection of vulnerabilities can be retrieved via the REST API. This can be done by

sending a request to the following base URL (Byers & Owen, 2019):

https://services.nvd.nist.gov/rest/json/cves/1.0

Additionally, specific result collections can be retrieved by adding query parameters (Byers & Owen, 2019).

At the time of writing, the prototype utilized two of the query parameters, namely **startIndex** and **resultsPerPage**. The **startIndex** allows setting off continuous requests by increasing the **startIndex** for every request to retrieve new data. The **resultsPerPage** parameter limits the maximum number of retrieved elements within a collection. This parameter has proved vital when connecting to the NVD. Although the page size is limited to 5000 entries, the prototype's **resultsPerPage** are limited to a value of 500. This is due to the fact that larger sizes e.g. 700 and 1000, in some cases results in too large data amounts, causing content to be cut off and yield an "unexpected end of file error". The size of 500 yielded complete results.

A query as it is utilized within the prototype is shown below. It retrieves the first 500 results.

https://services.nvd.nist.gov/rest/json/cves/1.0?resultsPerPage=500&startIndex=0

Of course, there are approximately 140.000 CVE entries. Consequently, the prototype iterates while incrementing the **startIndex** until all CVE entries are retrieved. To outline the process of retrieving data, the pseudo-code for the connection to the NVD is given in algorithm 1.

When taking a closer look at the algorithm one might notice the extra "while" loop if the connection could not be established as well as "wait" statements. These two clauses are required due to the configuration of the NIST firewall. In order to prohibit denial-of-service attacks, NIST limits the access to their database (Byers & Owen, 2019). As the documentation does not mention the amount of required waiting time between requests, a decision to wait 30 seconds when a connection cannot be established has been made. Moreover, there is a waiting time of 7 seconds before setting of a new request with increased startIndex. This way it is feasible to retrieve all the data from the NVD at once.

Another important aspect to consider is that during the prototype development, the downloaded data was first stored in a file which was later used as input for the application. The reason for this was to reduce network traffic and speed up application development while accessing the data locally. Furthermore, having

Algorithm 1: Retrieving data from NVD

```
begin
    initialize startIndex;
    initial query to get total results;
    while there are more results do
        create request with new startIndex;
        while connection not established do
            try to establish a connection;
            if connection successful then
                store data;
                break;
            else
                wait 30s;
        increment startIndex;
        wait 7s;
    store data in file;
```

static files simplified the analysis of the contents in terms of what was offered and how data could be accessed. Additionally, it enabled extracting the data for later training of the classifiers.

In the future this static process will change to a more dynamic approach where only the newest data will be queried utilizing a modification date within the requests which is supported by the NVD. More specifically, these parameters are called **modStartDate** and **modEndDate**. Hence, specific entries before, after or in between certain modification dates can be accessed. A query could look like the following:

```
https://services.nvd.nist.gov/rest/json/cves/1.0?
modStartDate=2019-12-31T00:00:00:000 UTC-00:00
```

The spring framework provides the libraries for establishing a connection to NVD's REST API. The utilized classes are:

- **org.springframework.web.client.RestTemplate** - to establish a connection, retrieve the data in JSON format and map it to generated classes
- **org.springframework.web.util.UriComponentsBuilder** - to build the URL with its query parameters

Due to the fact that the NVD provided its own custom format, a representation of the JSON file had to be created as java classes. Therefore, the idea was to use a class generator called **org.jsonschema2pojo** to create classes from a JSON schema. Although there is a schema[22] provided by the NVD, it did not represent the JSON data returned by the NVD REST API at the time of writing. Consequently, the required classes were directly generated from the JSON file which contained all NVD entries. This way it was feasible to generate classes without the need of programming the exact representation by hand. It enabled the mapping of the JSON file to java objects, thus, simplifying the latter mapping to the custom format which was described in section 4.3.2.

The data retrieval for Packetstorm proved to be simpler as it did not require any waiting time or mapping to an intermediate format. It will be described in the next section.

5.1.1.2 Connecting to Packetstorm

In contrast to NVD, Packetstorm does not provide a RESTful interface to access resources in XML or JSON format. It works with HTML webpages. Consequently, an approach to generate classes representing intermediate objects is not applicable. Although Packetstorm offers data within an RSS feed, it does not support historic data. Therefore, an analysis of the returned HTML pages has been conducted and relevant contents have been isolated.

As there is no structured information within a Packetstorm entry, the description and the published date have been considered as the most important aspects. Figure 30 shows an excerpt of an HTML page in Packetstorm. Every <dl> tag with **class=”file”** represents an entry on the web page. The nested objects with **class=”detail”** hold the vulnerability description. Knowing this, it is feasible to loop through all entries on the webpage and extract their data.

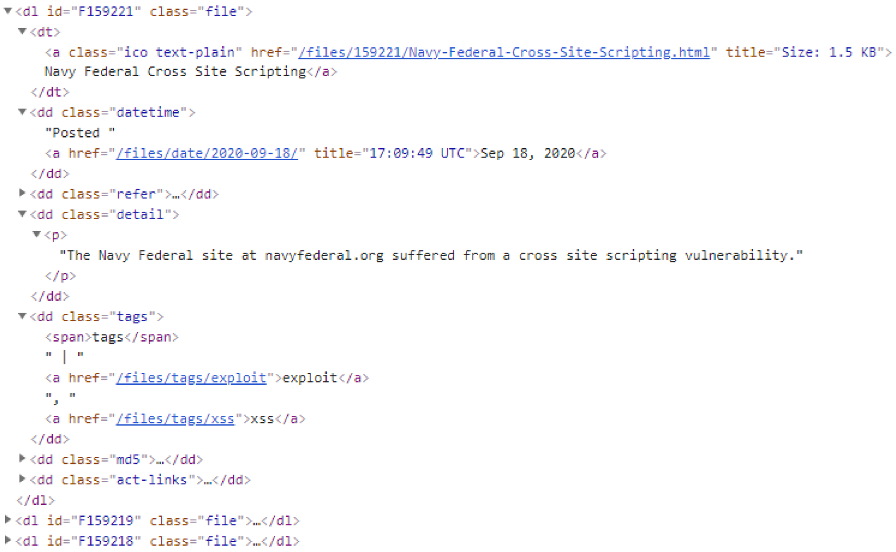
For the purposes of this master thesis only the exploit section of Packetstorm was considered. Other categories in Packetstorm are advisories, tools and whitepapers. However, they have not been regarded as relevant as advisories are mostly linked to CVE entries which are already covered by connecting to the NVD. Moreover, tool promotion and papers do not lie in the focus of this thesis.

As Packetstorm provides its data on webpages, there is a default limit of 25 entries per page. Consequently, navigation between those pages is required. This can be done by accessing the following URL and incrementing the <**pageNumber**> starting from 1.

https://packetstormsecurity.com/files/tags/exploit/page<**pageNumber**>

The connection to Packetstorm as well as the data retrieval process is outlined in algorithm 2.

[22] https://csrc.nist.gov/schema/nvd/feed/1.1/nvd_cve_feed_json_1.1.schema



```
▼<dl id="F159221" class="file">
  ▼<dt>
      <a class="ico text-plain" href="/files/159221/Navy-Federal-Cross-Site-Scripting.html" title="Size: 1.5 KB">
      Navy Federal Cross Site Scripting</a>
    </dt>
  ▼<dd class="datetime">
      "Posted "
      <a href="/files/date/2020-09-18/" title="17:09:49 UTC">Sep 18, 2020</a>
    </dd>
  ▶<dd class="refer">…</dd>
  ▼<dd class="detail">
    ▼<p>
        "The Navy Federal site at navyfederal.org suffered from a cross site scripting vulnerability."
      </p>
    </dd>
  ▼<dd class="tags">
      <span>tags</span>
      " | "
      <a href="/files/tags/exploit">exploit</a>
      ", "
      <a href="/files/tags/xss">xss</a>
    </dd>
  ▶<dd class="md5">…</dd>
  ▶<dd class="act-links">…</dd>
  </dl>
▶<dl id="F159219" class="file">…</dl>
▶<dl id="F159218" class="file">…</dl>
```

Fig. 30. HTML text of a Packetstorm webpage

Algorithm 2: Retrieving data from Packetstorm

```
begin
    initialize startIndex;
    while some stop condition do
        create request with new startIndex;
        retrieve HTML page;
        parse HTML page;
        get body of HTML page;
        find all elements with class ”file”;
        for all elements do
            get child with class ”detail” from the element;
            store data;
        increment startIndex;
    store data in file;
```

For this purpose, the spring framework is used to connect to Packetstorm. Furthermore, a webcrawler contained in the Jsoup library is utilized to parse the HTML. Therefore, the Packetstorm adapter makes use of the following classes:

- **org.springframework.web.client.RestTemplate** - to establish a connection and retrieve the HTML page
- **org.springframework.web.util.UriComponentsBuilder** - to build the URL with its query parameters
- **org.jsoup.Jsoup** - to parse the retrived HTML into a document
- **org.jsoup.nodes.Document** - the document representation of the HTML file
- **org.jsoup.nodes.Element** - An HTML element containing tags, attributes and child nodes
- **org.jsoup.nodes.Elements** - A list of elements. Relevant when filtering for the class "file" within the document

In contrast to the data contained in the NVD, the content of Packetstorm is unstructured and no XML or JSON schema is provided. Thus, there is no intermediate format for storage. However, only the description of an entry in Packetstorm is relevant and consequently, no additional structure is required before undergoing the Named Entity Recognition step. This leads us to the next section, the training of the named entity extractors.

5.1.2 Training of the Named Entity Extractors

Before applying a named entity extractor to plain text, it is necessary to train a classifier. This requires the annotation of documents in order to form a training and a test set. For this annotation step the tool doccano (Nakayama, Kubo, Kamura, Taniguchi, & Liang, 2018) with the sequence labelling configuration has proved to be useful. It provides a neat user interface enabling the creation of labels beforehand and assigning shortcuts to each of them which allows for a rapid annotation process.

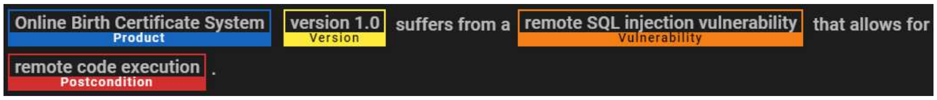
Also other approaches to annotation were tested out, namely ".tsv" files inside Excel or the BRAT[23] rapid annotation tool. Annotating with ".tsv" files was considered useful only when annotating smaller sets.
It is easy to lose track of the sentence structure due to the fact that it requires tokenized sentences that are annotated word by word where each words presents a row. Although BRAT is an established tool, doccano has been chosen over it due to the fact that it provides a more appealing user interface and user interaction in general. Moreover, doccano allows the import of a single file that is split line by line instead of requiring multiple individual files as input.

As specified in section 4.4.5 the labels utilized for classification are **Product**,

[23] `https://brat.nlplab.org/about.html`

Version, **Vulnerability**, **Precondition** and **Postcondition**. Four examples of annotated documents are contained in figures 31, 32, 33 and 34.

Figure 31 shows a standard entry within Packetstorm. It contains a product, version, vulnerability and postcondition.

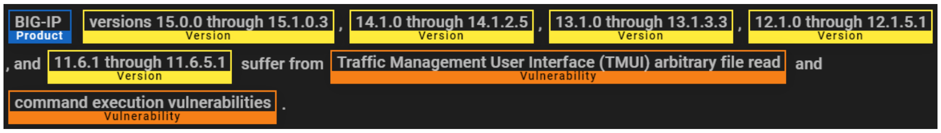


Online Birth Certificate System version 1.0 suffers from a remote SQL injection vulnerability that allows for remote code execution .

Fig. 31. Annotated example from Packetstorm

While product and version are straightforward and usually represented in the same structure of a product followed by its version, vulnerability and postcondition require more specification. In case of entries within Packetstorm the vulnerability is often introduced with a ”suffers from” and ended with ”vulnerability”. Postconditions are in most cases preceded by an ”allows for”. So whenever there is a ”suffers from”, the sequence is labelled as vulnerability, while sequences with a previous ”allows for” are categorized as postcondition. The chosen formulations are only exemplary and documents are not limited to this wording. However, a large amount of documents is based on these.

The next example as shown in figure 32 contains multiple versions as well as multiple vulnerabilities.

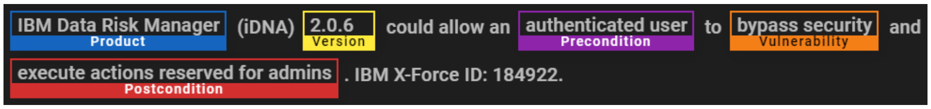


BIG-IP versions 15.0.0 through 15.1.0.3 , 14.1.0 through 14.1.2.5 , 13.1.0 through 13.1.3.3 , 12.1.0 through 12.1.5.1 , and 11.6.1 through 11.6.5.1 suffer from Traffic Management User Interface (TMUI) arbitrary file read and command execution vulnerabilities .

Fig. 32. Annotated example from Packetstorm with multiple versions

As before, the structure is product followed by version. In this case multiple versions including ranges are contained within the entry. Consequently, we have defined a guideline to combine ranges inside one single version label that can later be split. However, additional ranges are labelled individually. Moreover, when there are multiple vulnerabilities, multiple labels are provided.

As Packetstorm rarely provides preconditions, the following two examples focus on entries regarding the CVE/NVD. Figure 33 illustrates a vulnerability containing a precondition.

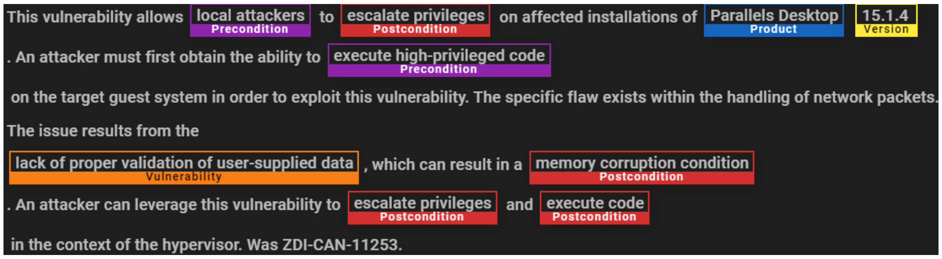


IBM Data Risk Manager (iDNA) 2.0.6 could allow an authenticated user to bypass security and execute actions reserved for admins . IBM X-Force ID: 184922.

Fig. 33. Annotated example from NVD including precondition

While product and version are provided in the same structure as before, additional information can be gained here. The "authenticated user" indicates that authentication is required in order to exploit the vulnerability. Moreover, in contrast to Packetstorm CVE/NVD entries usually do not contain the term "suffers from". Vulnerabilities within the text can come in various forms. In this case the "bypass security" vulnerability is preceded by "could allow an authenticated user to". Furthermore the postcondition that results from the security bypass is given after the "and".

The next example as provided in figure 34 describes a different formulation.



This vulnerability allows local attackers to escalate privileges on affected installations of Parallels Desktop 15.1.4 . An attacker must first obtain the ability to execute high-privileged code on the target guest system in order to exploit this vulnerability. The specific flaw exists within the handling of network packets. The issue results from the lack of proper validation of user-supplied data , which can result in a memory corruption condition . An attacker can leverage this vulnerability to escalate privileges and execute code in the context of the hypervisor. Was ZDI-CAN-11253.

Fig. 34. Annotated example from NVD with different formulation

As before the text requires further understanding in order to annotate the entities correctly. In general, wordings like "local attacker", "remote attacker" or "authenticated user" refer to preconditions. So these often but not always refer to specific word sequences instead of being preceded by specific formulations. However, in case of the example shown in figure 34 - "an attacker must first obtain the ability to" - definitely indicates a precondition. A formulation such as "results from" indicates a vulnerability in the system. Whereas, "results in" or "can leverage" introduces postconditions.

The examples above illustrate considerations for the annotation process. It is important to mention here that entries may also be phrased differently and that not all word constructs can be included here. The text has been annotated to the best of our knowledge and reviewed.

Once the annotation had been fully conducted, the file was exported as JSON format. However, in order to train a CRF classifier contained within the Stanford NLP library, a conversion to ".tsv" format was required. Therefore, the text was programmatically tokenized and already created labels were assigned word by word. A complete list of all the annotated documents, as well as the program code can be found on the DVD whose content is explained in annex C. For later evaluation three CRF-based named entity extractors were trained based on the annotated files. One extractor for Packetstorm, one for NVD and one containing training data from both sources were implemented. The results in terms of precision, recall and F1 measure will be evaluated in section 5.2.

After discussing the approach for labelling the named entities, the following section will elaborate on how the output of the resulting named entity extractors has been included into the final application.

5.1.3 Integrating the Named Entity Extractors

Now that the annotation process has been discussed, this section will give insight into how the extracted information is utilized within the prototype.

The extracted information is mapped to the format described in section 4.3.2. NVD entries are already structured and all the information except for pre- and postcondition can be directly extracted. But Packetstorm provides information only in unstructured form. Therefore, in context of Packetstorm all applicable attributes within the model are filled, based on the results of the entity extractor.

This is important especially when it comes to the versions of a specific product that are affected by a vulnerability. The NVD provides this information as CPE entries and additionally contains ranges by specifying "versionStartIncluding", "versionEndIncluding" and "versionEndExcluding" (Byers & Owen, 2019). Packetstorm in contrast, requires this information to be extracted from text. In terms of a single version this is not an issue, as it can be directly specified after the extraction step. However, a little bit more effort is required when ranges of products are included. Examples and mappings are given below:

- versions 15.0.0 through 15.1.0.3 - can be mapped to the model as "startIncluding" with a value of 15.0.0 and "endIncluding" with a value of 15.1.0.3
- prior to 9.2.1 - can be mapped as "startIncluding" with a value of "*" and "endExcluding" with a value of 9.2.1

As depicted above, entity extraction alone is not enough in terms of versions as they may require additional processing in order to be mapped to the model.

Pre- and postcondition as well as the vulnerability are also extracted and displayed as text, enabling a more compact description. Although NVD already provides information in the form of the CWE, the vulnerability is extracted from text anyway to give further insight into the issue.

As not all information needs to be mapped from both data-sources, the adapter pattern helps to create two different processes to achieve this aim. The entity extraction step is not conducted within the main programme, but within the adapters to allow for two different CRF classifiers to be used. Only their output in the form of the vulnerability model is fed back into the main programme where the information is displayed and may be adapted by the end-user.

The resulting information can then be utilized to formulate a rule. This topic will be covered in the next section.

5.1.4 Generating Rule Text

Section 3.4 already gave an explanation of what a rule must contain as a whole and that this information is extracted partly from text and partly from structured information sources and then fed into a rule. But for this information to be revealed after an analysis, the rule text (anti-pattern) must be specified. Some examples for component-based rules that can already be represented were discussed in section 4.4.6. This section will provide some additions.

When considering an entry from Packetstorm, we can extract the product and the version which are the requirements for generating the rule text that will be evaluated against the system model during an analysis. All this enables the identification of vulnerable system components. On the one hand, when only individual versions are considered vulnerable this task is quite simple as only the specific versions need to be specified. On the other hand, ranges are currently not supported when conducting an analysis. Consequently, it is necessary to introduce new keywords to the grammar. Proposals for such keywords are discussed below. The following values for ”startIncluding”, ”endIncluding” and ”endExcluding” refer to the vulnerability model which was explained in section 4.3.2.

In case of affected products where the values ”startIncluding” and ”endIncluding” are specified, the keyword **”BETWEEN”** could be used, to allow the formulation of a rule text with regard to such a range. A corresponding example is shown below:

```
Type(”ANY”).tv(product = ”BIG-IP” AND (version BETWEEN(”15.0.0”,
”15.1.0.3”) OR version BETWEEN(”14.1.0”, ”14.1.2.5”)))
```

The structure inside **”BETWEEN”** allows for specifying lower and upper bounds of version ranges.

When it comes to affected products containing ”startIncluding” and ”endExcluding”, the keyword could be **”BETWEENEX”**. An example would be analogous to the one above which is why it will be omitted here. The difference to **”BETWEEN”** is that **”BETWEENEX”** defines a lower bound that is included in the analysis but the upper bound presents a version where the vulnerability has already been solved.

In other cases the version may be defined as ”prior to...” or ”before ...”. For this, the keyword **”BEFORE”** could be utilized. Such a configuration is indicated when the ”version” or ”startIncluding” are specified as ”*” depicting that any version version is affected, accompanied by an ”endExcluding”. An example is given below:

```
Type(”ANY”).tv(product = ”McAfee Web Gateway” AND (version
BEFORE(”9.2.1”)))
```

Additionally, it is also possible that the ”endIncluding” value is specified, which depicts that all previous versions including the specified version are affected by a vulnerability. This could be represented with the **”UNTIL”** keyword and is analogous to the example above.

In contrast to zero-day vulnerabilities as it is the case for rules utilizing the keyword **”BEFORE”** or **”UNTIL”**, vulnerabilities may be introduced in a certain version of software or hardware. If there is no available fix yet, then only the value for ”startIncluding” is set. This basically means that all versions starting from the specified entry are affected. Therefore, the keyword **”FROM”** could be a possible way for representation. An example rule for this case is displayed below:

```
Type(”ANY”).tv(product = ”Linux linux_kernel” AND (version
FROM(”4.8”)))
```

These representations are not yet possible. Moreover, it currently not feasible at all to include ranges without adding every possible version one by one. However, ranges could be integrated in the future, enabling a simpler representation. As far as the implementation of the prototype is concerned, the creation of rules is based on these newly introduced keywords and done programmatically.

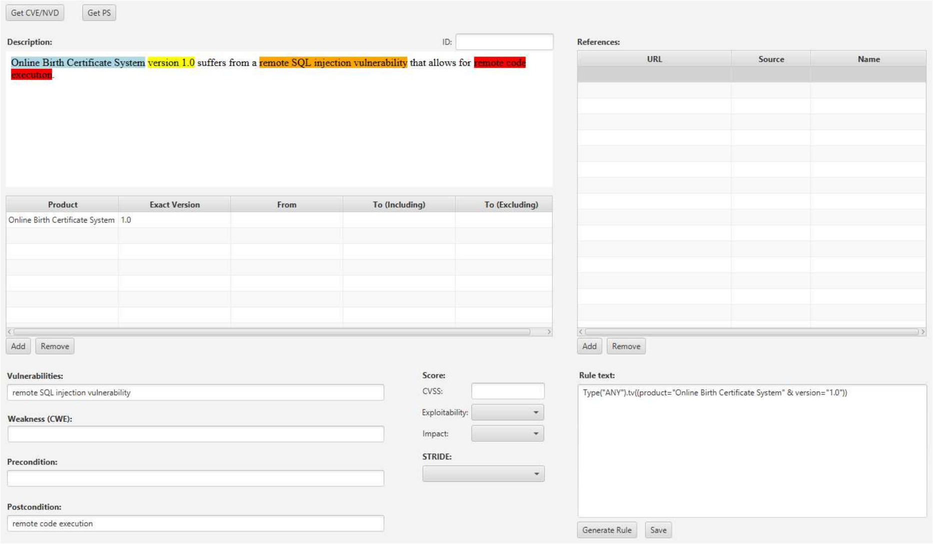
In order to validate the data that is extracted and to create valid rule text, the following section introduces a GUI enabling a validation process.

5.1.5 Validating the Information

As even well-trained NER systems yield incorrect results, there is always a certain error rate when developing software in the area of natural language processing. The application developed here is relevant to the security domain. Consequently, the results should be as correct as possible in order to identify existing security threats. Therefore, the error should be as low as possible in order to omit the incorrectly identified threats which the system actually does not suffer from. More importantly, threats that are incorrectly not identified although they are relevant to the system must be kept at a minimum.

Sentences might be formulated differently or contain special characters which the extractor does not recognize. Possibly, they were previously unknown to the extractor as some special cases were not included within the training set. Or they may have been included in the training set, but to an extent that their weight is too low to be identified correctly. Furthermore, word sequences could be incorrectly categorized as certain named entities although they are not relevant which might produce wrong results.

For these reasons a graphical user interface has been developed to give the end user insight into the rules to be generated and to enable a revision of the extracted data. The complete GUI is shown in figure 35. This GUI reflects every-



Description:

Online Birth Certificate System version 1.0 suffers from a remote SQL injection vulnerability that allows for remote code execution.

Product	Exact Version	From	To (Including)	To (Excluding)
Online Birth Certificate System	1.0			

URL	Source	Name

Vulnerabilities: remote SQL injection vulnerability

Weakness (CWE):

Precondition:

Postcondition: remote code execution

Rule text:

```
Type("ANY").tv((product="Online Birth Certificate System" & version="1.0"))
```

Fig. 35. Complete GUI

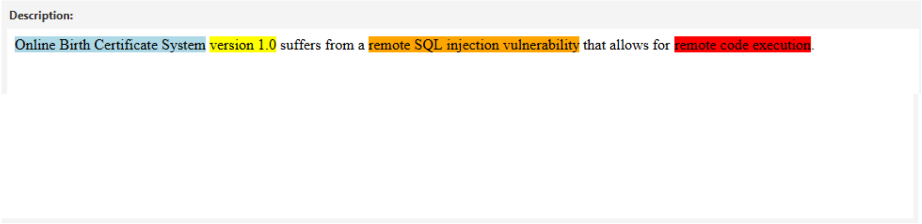
thing that that is required for storing vulnerabilities and has been specified so far:

- the **Description**
- the **ID** taken from the online resource (if available)
- the **Product(s)**
  NOTE: The columns inside the GUI were renamed for better understanding. The ”Exact Version” represents a specific version of a product. ”From” depicts ”startIncluding”, ”To (Including)” stands for ”endIncluding” and ”To (Excluding)” represents ”endExcluding”.
- the **References** (if available)
- the identified **Vulnerabilities**
- the **Weakness** (if available)
- the **Precondition** (if available)
- the **Postcondition** (if available)
- the **CVSS** score mapped to the allowed risk values (if available)
- the **STRIDE** category (if available)
- the **Rule Text** corresponding to the vulnerability

By selecting one of the two buttons **Get CVE/NVD** or **Get PS** the end user can select the data source to generate rules from. In the following, each part of the GUI will be explained in detail.

5.1.5.1 Describing the GUI for a Packetstorm Entry

The **description** field is simply an HTML text viewer. After running entity extraction on the provided text, the resulting named entities are matched again with the text which is afterwards colorized utilizing the HTML style attribute. An example is given in figure 36.



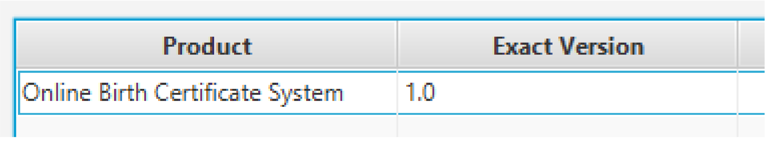
Description:

Online Birth Certificate System version 1.0 suffers from a remote SQL injection vulnerability that allows for remote code execution.

Fig. 36. Description field of the GUI

For each type of entity a different color was chosen in order to highlight the original text. This step was inspired by the Stanford NER programme (Finkel et al., 2005) which was used for the experiments before integrating the workflow into the application. **Blue** color depicts the product category, yellow the version and **orange** the vulnerability. Moreover, a precondition is marked **green** (this will be shown in the NVD example) and the postcondition **red**.

The **Affected Product** table lists all affected products by their names. In case of a CPE entry, only the vendor and the product name are listed instead of the whole CPE string. The table gives an insight into the versions that are affected. As far as Packetstorm is concerned, the information represented here is extracted from text. In terms of NVD the data is taken from the structured JSON file. Figure 37 shows an example entry.



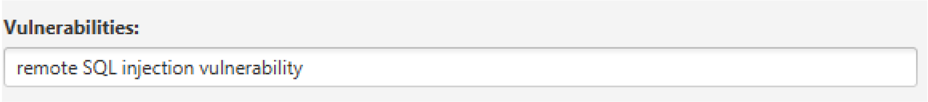
Product	Exact Version
Online Birth Certificate System	1.0

Fig. 37. Affected products table of the GUI

The affected product represented in the figure could be only one specific version as represented here. However, sometimes also ranges of product versions are specified. This case will be shown in section 5.1.5.2.

As Named Entity Recognition does not always find every entity or may classify incorrectly, two buttons were added to the product table. With the **Add** button, new rows may be added. The **Remove** button on the contrary can be used to delete rows that were incorrectly or accidentally added.

The **Vulnerabilities** field represents the vulnerabilities identified in the description text.



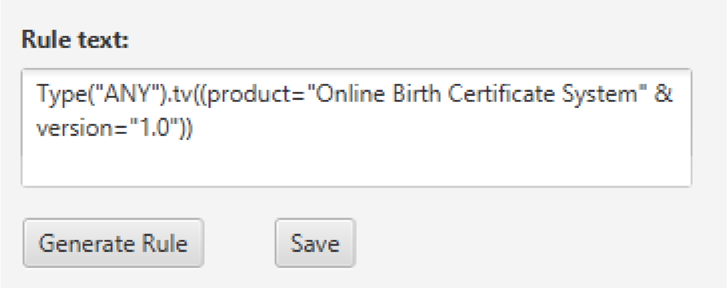
Vulnerabilities:

remote SQL injection vulnerability

Fig. 38. Vulnerabilities field of the GUI

In case the named entity recognizer does not find all vulnerabilities within the text, additional vulnerabilities may be added manually and can be separated using a semicolon ";".

The final field relevant for Packetstorm entries is the **Rule Text** field. It is automatically filled when the next entry is loaded. By clicking the **Generate Rule** button, the user may update the displayed rule if there are any changes inside the other fields of the GUI. An image of the respective field is shown in figure 39. **Save** stores the content shown in the GUI to the database.



Rule text:

Type("ANY").tv((product="Online Birth Certificate System" & version="1.0"))

Fig. 39. Rule text field of the GUI

The generated rules consist of the data that is represented in the **Affected Products** table and serves to identify products when running the analysis.

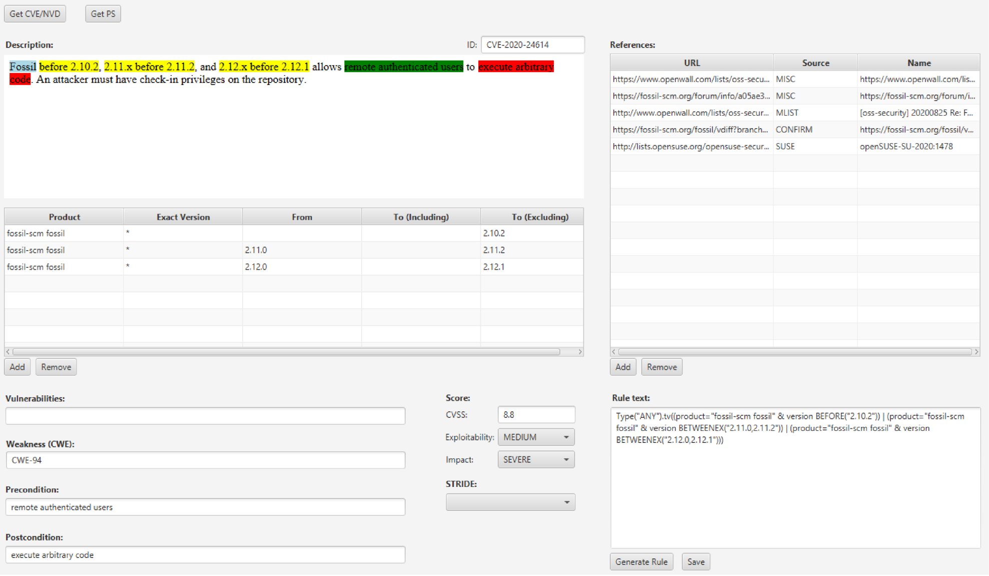
This section described the GUI in terms of entries retrieved from Packetstorm. However, the attentive reader may notice that not all of the GUI elements have been explained in detail yet. This is due to the reason that Packetstorm does not provide all the data that can be displayed within the GUI as it only satisfies the minimum requirements for representing the vulnerabilities. Therefore, the next section will provide a listing of the GUI elements utilized within a CVE/NVD entry.

5.1.5.2 Describing the GUI for an NVD Entry

Though the CVE/NVD entries contain the product name as well as the version inside the description, they are not included within the product table as these attributes can be extracted from JSON format. However, if an entity is recognized it will be highlighted within the description field anyway. An example based on CVE-2020-24614 is shown in figure 40.

What has been considered more important in terms of CVE/NVD entries are the pre- and postconditions. Consequently, more emphasis has been put on entries containing these two attributes when adding them to the training and test sets.

As in the previous section, the **Description** field contains the highlighted de-



Description: ID: CVE-2020-24614

Fossil before 2.10.2, 2.11.x before 2.11.2, and 2.12.x before 2.12.1 allows remote authenticated users to execute arbitrary code. An attacker must have check-in privileges on the repository.

Product	Exact Version	From	To (Including)	To (Excluding)
fossil-scm fossil	*			2.10.2
fossil-scm fossil	*	2.11.0		2.11.2
fossil-scm fossil	*	2.12.0		2.12.1

References:

URL	Source	Name
https://www.openwall.com/lists/oss-secu...	MISC	https://www.openwall.com/lis...
https://fossil-scm.org/forum/info/a05ae3...	MISC	https://fossil-scm.org/forum/i...
http://www.openwall.com/lists/oss-secur...	MLIST	[oss-security] 20200825 Re: F...
https://fossil-scm.org/fossil/vdiff?branch...	CONFIRM	https://fossil-scm.org/fossil/v...
http://lists.opensuse.org/opensuse-secur...	SUSE	openSUSE-SU-2020:1478

Vulnerabilities:

Weakness (CWE): CWE-94

Precondition: remote authenticated users

Postcondition: execute arbitrary code

Score:
CVSS: 8.8
Exploitability: MEDIUM
Impact: SEVERE

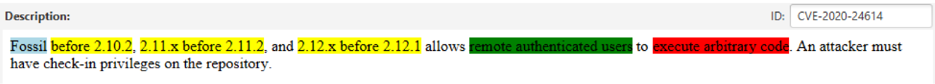
STRIDE:

Rule text:

```
Type("ANY").tv((product="fossil-scm fossil" & version BEFORE("2.10.2")) | (product="fossil-scm
fossil" & version BETWEENEX("2.11.0,2.11.2")) | (product="fossil-scm fossil" & version
BETWEENEX("2.12.0,2.12.1")))
```

Fig. 40. GUI in case of CVE-2020-24614

scription text of a CVE entry. An illustration is given in figure 41.



Description: ID: CVE-2020-24614

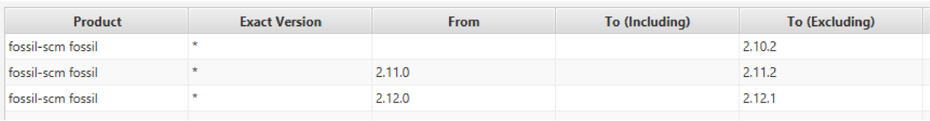
Fossil before 2.10.2, 2.11.x before 2.11.2, and 2.12.x before 2.12.1 allows remote authenticated users to execute arbitrary code. An attacker must have check-in privileges on the repository.

Fig. 41. Description of a CVE/NVD entry in the GUI

Additionally, the **ID** field is filled with the ID of the original CVE/NVD entry within the online repository in order to be able to uniquely identify the vulnerability in terms of updates inside the National Vulnerability Database.

In contrast to the table shown in the example of the previous section, this table contains ranges of product versions. Figure 42 contains a table regarding these ranges.

A range within product versions can, on the one hand, be represented with "*" for the **"exact version"** depicting all versions before a specific update. On the other hand, it may also start with **"from"** meaning that the vulnerability was introduced with a specific update of the product. As far as **"to (including)"**

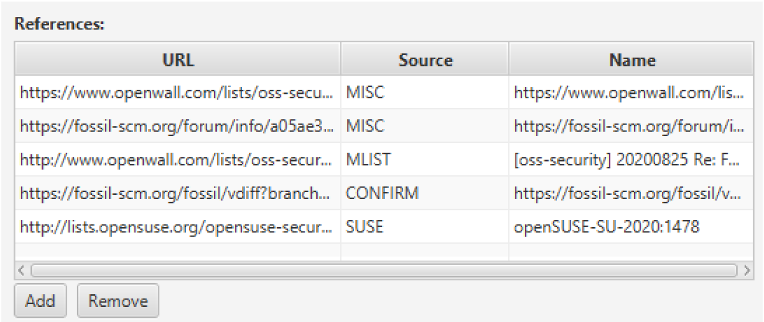


Product	Exact Version	From	To (Including)	To (Excluding)
fossil-scm fossil	*			2.10.2
fossil-scm fossil	*	2.11.0		2.11.2
fossil-scm fossil	*	2.12.0		2.12.1

Fig. 42. Affected products of the GUI with ranges

is concerned, product versions including this version are considered vulnerable. In case of **”to (excluding)”** all versions lower than the specified value are considered vulnerable.

The reference table provides links to advisories associated to the displayed vulnerability. This is exemplified in figure 43.



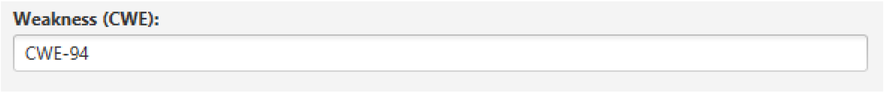
References:

URL	Source	Name
https://www.openwall.com/lists/oss-secu...	MISC	https://www.openwall.com/lis...
https://fossil-scm.org/forum/info/a05ae3...	MISC	https://fossil-scm.org/forum/i...
http://www.openwall.com/lists/oss-secur...	MLIST	[oss-security] 20200825 Re: F...
https://fossil-scm.org/fossil/vdiff?branch...	CONFIRM	https://fossil-scm.org/fossil/v...
http://lists.opensuse.org/opensuse-secur...	SUSE	openSUSE-SU-2020:1478

Fig. 43. References table of the GUI

As for the product table, experts can add a new row by clicking the **Add** button and remove an existing entry by clicking **Remove**.

In addition, the NVD associates generic weaknesses in most of its entries. These weaknesses reflect entries of the Common Weakness Enumeration (CWE). An example is given in figure 44.

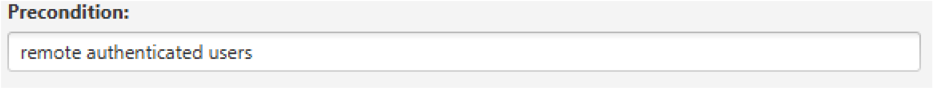


Weakness (CWE):
CWE-94

Fig. 44. Example of a CWE derived from an NVD entry

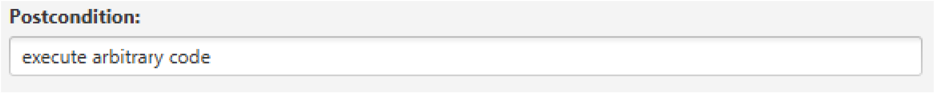
The representation of the CWE ID enables the user to search for the specific entry inside the CWE dictionary in order to find possible solutions and mitigation strategies.

Due to the fact that some vulnerability descriptions contain additional information, i.e. pre- and postcondition, they are also included in the rule creation process. Examples are given below in figures 45 and 46.


Precondition:

remote authenticated users

Fig. 45. Example for an extracted precondition of an NVD entry


Postcondition:

execute arbitrary code

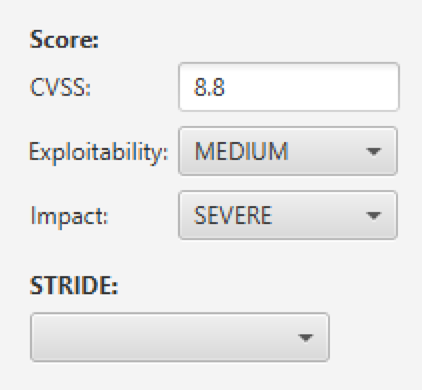
Fig. 46. Example of an extracted postcondition of an NVD entry

Although pre- and postcondition are not applicable at the moment, they will be able to leverage the capabilities of threat modelling in the future by firing rules only if the precondition has been fulfilled and will be using the postcondition as input for the next step within the analysis.

As the rules that are produced will contain scores, the CVSS as well as scores for impact and exploitability are reflected in the GUI. A representation is shown in figure 47.

The CVSS is a value taken directly from the JSON returned by the NVD REST API. Exploitability and impact are converted to the risk matrix values as specified in section 4.3.4. The value for STRIDE is derived utilizing the definition provided in section 4.3.5. Finally, the generated rule text is shown in figure 48.

The fact that a Packetstorm entry does not contain as much information as a CVE/NVD entry does not mean that a Packetstorm entry cannot have an associated CWE or a scoring. The GUI was not only designed to display the extracted information in order to revise extracted data. The intention behind its



Score:

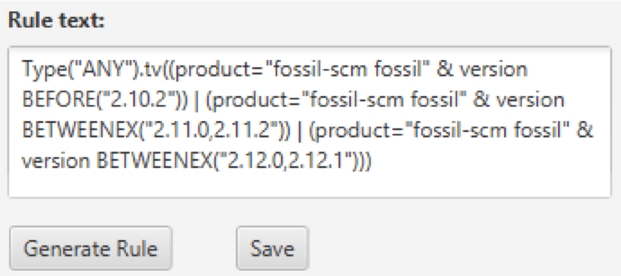
CVSS: 8.8

Exploitability: MEDIUM

Impact: SEVERE

STRIDE:

Fig. 47. Examples for CVSS, Exploitability and Impact Scores



Rule text:

```
Type("ANY").tv((product="fossil-scm fossil" & version
BEFORE("2.10.2")) | (product="fossil-scm fossil" & version
BETWEENEX("2.11.0,2.11.2")) | (product="fossil-scm fossil" &
version BETWEENEX("2.12.0,2.12.1")))
```

Fig. 48. An example of a generated rule

design is rather to manually enhance the displayed information with additional knowledge and to assign e.g. a CWE or values for exploitability, impact and the STRIDE category. This way data from resources without detailed information such as Packetstorm can become more sophisticated.

The validation process helps in correcting the resulting data and is necessary due to potential errors when identifying named entities.

5.2 Results

The previous sections have covered the architecture and implementation details underlying the prototype, as well as a description of the GUI. What has not been discussed so far are results that were achieved. Therefore, the following section will deal with some generated rules with regard to a more sophisticated version of the illustrative system model from figure 2. Afterwards measurements in terms of precision, recall and F_1measure will be interpreted.

5.2.1 Examples for Generated Rules

So far, it has only been possible to find generic vulnerabilities within system configurations. The newly integrated data and its respective rules enable the identification of product specific vulnerabilities. Therefore, assume that the illustrative system model has been extended with attributes depicting products that are running on the components. An analysis of the system including these newly generated rules is now capable of revealing further vulnerabilities in addition to the ones specified in section 3.2.3. Figure 49 shows an image of the system model including the product attributes.

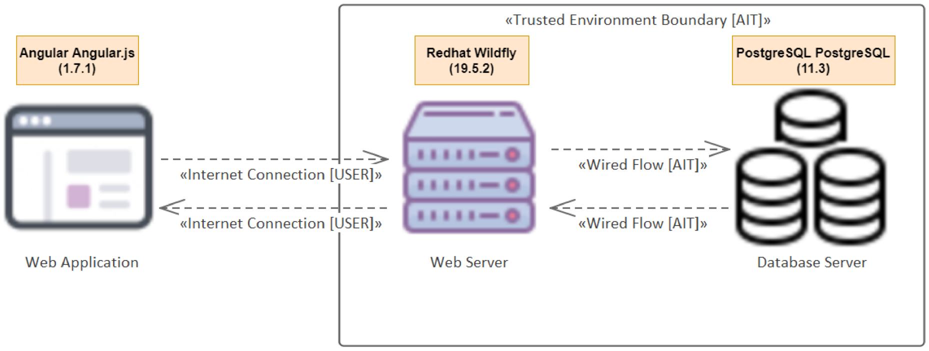

«Trusted Environment Boundary [AIT]»
Angular Angular.js (1.7.1)
Redhat Wildfly (19.5.2)
PostgreSQL PostgreSQL (11.3)
«Internet Connection [USER]»
«Internet Connection [USER]»
«Wired Flow [AIT]»
«Wired Flow [AIT]»
Web Application
Web Server
Database Server

Fig. 49. An illustrative system model with specified products and versions

It has been specified that the originally generic web application is an angular.js application in version 1.7.1. Moreover, the Web server is a Wildfly server in version 19.5.2. Finally, the database server is hosting a PostgreSQL database in version 11.3.

By analyzing this system configuration with some newly created rules, new vulnerabilities will be revealed. Examples are given in tables 15, 16 and 17.

Example 1	**Cross Site Scripting in angular.js**
Description	angular.js prior to 1.8.0 allows cross site scripting. The regex-based input HTML replacement may turn sanitized code into unsanitized one. Wrapping ”< option>” elements in ”<select>” ones changes parsing behavior, leading to possibly unsanitizing code.
Product	angular angular.js
EndExcluding	1.8.0
Vulnerabilities	cross site scripting
Weakness	CWE-79
Postcondition	possibly unsanitizing code
CVSS	5.4
Exploitability	MEDIUM
Impact	MODERATE
References	https://lists.apache.org/thread.html/rfa2b19d01d10a8637dc319a7d5994c3dbdb88c0a8f9a21533403577a@%3Cozone-issues.hadoop.apache.org%3E, https://snyk.io/vuln/SNYK-JS-ANGULAR-570058, ...
Generated Rule Text	Type(”ANY”).tv((product=”angularjs angular.js” & version BEFORE(”1.8.0”)))

Table 15. Example of a vulnerability about cross site scripting in angular.js taken from CVE-2020-7676

Table 15 displays a rule regarding the web application. As specified above, it is an angular.js 1.7.1 application. When taking a closer look at table 15 one can see that ”EndExcluding” is set to 1.8.0 and no other version attribute has been set. Therefore, our application is affected by this vulnerability due to the fact that every version **”BEFORE”** 1.8.0 suffers from cross site scripting. Moreover, the associated weakness is CWE-79: ”Improper Neutralization of Input During Web Page Generation (’Cross-site Scripting’)”, and the postcondition describes that an exploit may lead to an unsanitization of code. By looking up the exploitability and impact values inside the risk matrix specified in table 14 we can derive a severity of 3 which matches the medium CVSS value of 5.4. When considering risk treatment, this is a vulnerability which experts may want to fix.

Example 2	**Memory Disclosure in PostgreSQL**
Description	Postgresql, versions 11.x before 11.5, is vulnerable to a memory disclosure in cross-type comparison for hashed subplan.
Product	postgresql postgresql
StartIncluding	11.0
EndExcluding	11.5
Vulnerabilities	memory disclosure
Weakness	CWE-125, CWE-200
CVSS	2.2
Exploitability	LOW
Impact	MODERATE
STRIDE	INFORMATION DISCLOSURE
References	https://www.postgresql.org/about/news/1960/, https://bugzilla.redhat.com/show_bug.cgi?id=CVE-2019-10209
Generated Rule Text	Type(”ANY”).tv((product=”postgresql postgresql” & version BETWEENEX(”11.0,11.5”)))

Table 16. Example of a vulnerability about memory disclosure in PostgreSQL taken from CVE-2019-10209

In table 16 we can see a memory disclosure vulnerability affecting our PostgreSQL database server. In this case, the specified version 11.3 lies within the range of the vulnerable version 11.0 and 11.5, where 11.5 is excluded and, therefore, not considered vulnerable anymore. Consequently, a rule created from this input must utilize the suggested keyword **”BETWEENEX”** when checking for vulnerabilities of that range. Furthermore, this vulnerability has two CWEs assigned which provide further information to the experts. In general, this vulnerability is considered less severe as utilizing the risk matrix as lookup table yields a severity level of 2. It is well possible that an organization is willing to accept the resulting risk. However, depending on the application domain, as well as an organization’s objective it may also be fixed by applying mitigation measures.

The final example that is exemplified in table 17 is a deserialization of untrusted data in Wildfly. All versions up to 20.0.0 are affected. As our version 19.5.2 is included, our product lies within the range and is therefore vulnerable to deserialization of untrusted data (CWE-502). In this case the exploitability score is low while the impact score is classified as severe. This is why the matching risk matrix value sets a severity score of 4. Consequently, an organization will consider transferring that risk if possible. In case the component is necessary to reach their objective, it will mitigate that risk.

Example 3	**Deserialization of Untrusted Data in Wildfly**
Description	A vulnerability was found in Wildfly in versions before 20.0.0.Final, where a remote deserialization attack is possible in the Enterprise Application Beans(EJB) due to lack of validation/filtering capabilities in wildfly.
Product	redhat wildfly
EndExcluding	20.0.0
Weakness	CWE-502
CVSS	7.5
Exploitability	LOW
Impact	SEVERE
References	https://bugzilla.redhat.com/show_bug.cgi?id=CVE-2020-10740
Generated Rule Text	Type(”ANY”).tv((product=”redhat wildfly” & version BEFORE(”20.0.0”)))

Table 17. Example of a vulnerability about deserialization of untrusted data in Wildfly taken from CVE-2020-10740

The presented examples represent only a portion of potentially detected vulnerabilities. A server could for example run multiple web applications with a different basis (e.g. angular.js, vue.js). Moreover, the machine that the web-server is running on could be further specified. Maybe there is even other software running on the machine. Although this additional information might not be directly relevant for the built application itself, it is very well necessary to include everything that is known into the evaluation. An application that is not coupled to the system may still pose a threat in case it is exploited.

For simple identification of a vulnerability, a description and the product with its version attributes are sufficient. Nevertheless, the examples above include additional information about vulnerabilities that can inform decision and will have their impact on risk treatment.

This section discussed results of the rule generation utilizing the illustrative example from section 3. In the following, measurement results regarding the Named Entity Recognition task will be explained.

5.2.2 Measurement Results

Section 3.6.6 introduced means to measure (not) correctly identified entities. Therefore, this section will deal with the respective measures and compare the results of three approaches towards the named entity extraction task. In order to get statistically relevant results, the experiments were conducted 100 times. Moreover, the original sets were shuffled for every iteration and afterwards split into a training and a test set.

To omit the effect of outliers that appear only once in the whole dataset and are totally different in their structure in contrast to the other documents, they were removed from the respective datasets. Although these outliers would not have a drastically negative effect on the training set, the measurement results would be affected intensely if contained in the test set. E.g. one document in Packetstorm contained approximately 70 versions in a format different from any other document, that, if not contained in the training set would boost the number of false negatives extremely.

The training set size has been defined to be 70% of the original set which served for training the named entity extractor. Consequently, the resulting entity extractor has been tested against a set with a size of 30% of the original set. Precision, recall and F_1measure will be displayed for each of the five categories: product, version, precondition, postcondition and vulnerability. Moreover, the number of true positives, false positives and false negatives will be illustrated as well. Please note that the values in the following tables are rounded.

5.2.2.1 Results of the Packetstorm Named Entity Extractor

The first entity extractor that will be discussed here is the one consisting only of entries from Packetstorm. A total of 238 entries were annotated. For each iteration 167 annotated documents were put into the training set. The test set consists of 71 entries. The measurement results are contained in table 18.

	Precision	Recall	F_1measure	TP	FP	FN
Product	0.964	0.912	0.937	65.87	2.47	6.38
Version	0.943	0.887	0.913	60.82	3.66	7.85
Precondition	0.0	0.0	0.0	0	0.053	2.94
Postcondition	0.689	0.416	0.504	3.72	1.8	5.46
Vulnerability	0.856	0.834	0.845	66.54	11.26	13.33
Total	0.911	0.847	0.878	196.95	19.24	35.78

Table 18. Measurements for the Packetstorm classifier with 238 entries

In terms of the categorization of a product, the results seem quite promising. The precision tells us about the accuracy of the detected product entities. It has a value of 96.4% and is, consequently, very high. Moreover, the recall concerning the product is an excellent outcome as well, as it measures all correctly identified entities by all that should have been identified. 91.2% of the entries that should be labelled were correctly categorized. As for the weighted average of precision and recall, namely the F_1measure, the resulting value is 93.7%.

When taking a look at the identified version, the values for precision and recall are also favorable. Although the results are not as good as for the product, they are still quite close to each other. This probably results from the fact that a large amount of Packetstorm entries start with the product name accompanied by the specification of the version. Furthermore, the structure within entries is quite similar.

The poor outcome for the precondition entity can be attributed to the fact that the entries in Packetstorm contain only a marginal quantity of preconditions. Consequently, it is unlikely that a random training set contains enough entries to correctly identify preconditions. Moreover, the test set must also contain entries including a precondition to be detected.

In terms of the postcondition, the extractor produced much better but still only mediocre results. As far as the precision is concerned it has a fair outcome of 68.9%. However, a recall of 41.6% is still pretty low. When taking a look at the F_1measure, a little bit more than 50% is not a very good result and shows us that precision and recall are not very well balanced in this case. But as for the precondition, the amount of postconditions included in Packetstorm entries was still relatively low.

A more encouraging result is the vulnerability category. The precision as well as the recall are located in the low to mid 80 percentages, which results from the fact that vulnerabilities often come in the form of a word sequence ending with "vulnerability".

In general, the total results are highly satisfactory. An overall precision of 91.1% tells us that this amount of identified entities is categorized correctly. The somewhat lower result for the recall of 84.7% is still satisfying, as human interaction is required for validation of the resulting rules anyway. Even the F_1measure is quite high with it being 87.8% which tells us that precision as well as recall have provided good results.

The homogeneous structure of most of the Packetstorm entries is an important contributor to the results shown in table 18. In contrast, the low rate of annotated entities regarding pre- and postconditions yielded mediocre performance. To enhance the performance of the classifier, more training data with special regard to pre- and postconditions would be needed.

This section has dealt with the extractor trained on Packetstorm entries only. Therefore, the next section will describe the results of the NVD based extractor.

5.2.2.2 Results of the NVD Named Entity Extractor

As far as the NVD NER classifier is concerned, a total of 244 entries were annotated. As before the entries were shuffled and then divided into a training and test set. The training set consists of 171 documents, while the test set contains 73 NVD entries. Table 19 contains the results of the measurements.

	Precision	Recall	F_1measure	TP	FP	FN
Product	0.861	0.732	0.791	65.68	10.73	24.32
Version	0.878	0.814	0.844	72.47	10.03	16.58
Precondition	0.819	0.662	0.731	27.86	6.15	14.29
Postcondition	0.641	0.543	0.587	36.71	20.46	31.13
Vulnerability	0.760	0.598	0.669	51.75	16.4	34.72
Total	0.800	0.678	0.734	254.47	63.77	121.04

Table 19. Measurements for the NVD classifier with 244 entries

Again, the precision for product and the version are fairly high, due to the similarity in the word sequence. Nonetheless, the results of the NVD extractor are a bit worse than for the Packetstorm classifier. As CVE descriptions inside the NVD utilize various different formulations, the product and version names are not always mentioned at the beginning of an entry. In many cases they are located somewhere in the middle of a sentence. Therefore, the word sequence itself is an indicator, but in contrast to Packetstorm the positioning inside the sentence as well as the context surrounding this sequence may be arbitrary. This makes it more complicated as a product may consist of an arbitrary number of words where sometimes even the user is unsure of how to categorize this information. The arbitrary position as well as long product names are reasons why the recall for a product here is much lower than for the Packetstorm extractor. Considering the heterogeneity of the data F_1measures of 79.1% and 84.4% are satisfactory results.

In terms of the detection of preconditions, the NVD based classifier outperforms the Packetstorm extractor. 81.9% of the classified entities are classified correctly. However, in comparison the recall for the precondition is rather low due to the high amount of false negatives in relation to the amount of true positives.

Assigning a postcondition or a vulnerability label can be quite cumbersome, for the human annotator as well as for the classifier. Although a precision of 76% for the vulnerability label is generally not a bad result, the result of 64.1% for the postcondition is far from optimal. The reason behind this is that postconditions and vulnerabilities themselves often use the same wording which is adapted in the classifier and introduces ambiguity. Therefore, assigning the correct label is strongly correlated to the context and the surrounding word sequences. This is also why the recall for both of these measures is below 60% and why the number of false positives for the precondition is high. To counter this, a larger training set would be required to trim the classifier to certain contexts instead of relying too much on simple wording.

The total results are acceptable but could be better. Moreover, the heterogeneity within the training data affects the resulting classifier. In the future a larger training set could be used in order to further enhance the performance of the named entity extractor and to reduce the effect of ambiguity.

The last two sections have dealt with entity extractors specific to the domains of Packetstorm and the NVD. In order to evaluate synergies between entity extractors, a combined classifier has been developed based on both data sets.

5.2.2.3 Results of a Combined Named Entity Extractor

The combined entity extractor has been trained and tested based on 238 documents from Packetstorm and 244 from the NVD. This makes a total of 482 entries. For the experiments, both datasets were merged and randomized. From this a training set of 338 and a test set of 144 were created. As before, measurements in terms of precision and recall were conducted. The results are listed in table 20.

	Precision	Recall	F_1measure	TP	FP	FN
Product	0.889	0.817	0.852	131.66	16.48	29.6
Version	0.904	0.856	0.880	133.2	14.14	22.27
Precondition	0.803	0.614	0.694	27.1	6.57	17.29
Postcondition	0.633	0.544	0.584	42.11	24.63	35.49
Vulnerability	0.810	0.715	0.759	119.1	28.19	47.46
Total	0.835	0.750	0.790	453.17	90.01	152.11

Table 20. Measurements for the combined classifier with 482 entries

On the one hand, the extractor produces worse results in terms of product and version as far as Packetstorm is concerned. On the other hand, the precision and the recall for both categories are better than in the original NVD extractor. The reason for this behaviour is the fact that product and version in NVD entries differ from their equivalents in Packetstorm due to different surrounding contexts.

The precision for the precondition of 80.3% for the combined extractor is very close to the precision of the original NVD extractor with 81.9%. However, the recall is a bit lower. The good precision being almost the same as for the NVD extractor is due to the amount of preconditions contained in the training data for the NVD. In contrast to that, Packetstorm hardly contained any preconditions. Consequently, this result is closely related to the NVD classifier.

When considering the ambiguity of postcondition and vulnerability based on the wording, the result for vulnerability lies somewhere in the middle between the two distinct extractors while the detection of the postconditions is worse.

It may seem that the combined classifier with a total precision of 83.5% and a recall of 75% performs better than the extractor specifically tailored to CVE/NVD entries. But the results here do not represent a test against only CVE/NVD entries but a test against the combined NVD and Packetstorm documents. Consequently, the combined named entity extractor should be directly tested against the test set from Packetstorm and the test set derived from the NVD.

5.2.2.4 Applying the Combined Named Entity Extractor to Packetstorm and NVD Test Sets

For the following experiments the combined named entity extractor was trained on 50% Packetstorm and 50% CVE/NVD data, in contrast to a completely randomized training. More precisely, it was trained on exactly 167 Packetstorm documents and respectively 171 from the NVD. Nonetheless, the experiment was conducted 100 times with varying training and test sets for each of the two data sources. The combined extractor was then applied to test sets for Packetstorm and CVE/NVD documents separately. The results are displayed in tables 21 and 22.

	Precision	Recall	F_1measure	TP	FP	FN
Product	0.941	0.912	0.926	66.04	4.14	6.35
Version	0.936	0.896	0.915	60.93	4.14	7.2
Precondition	0.054	0.088	0.031	0.08	0.97	2.53
Postcondition	0.525	0.588	0.544	5.33	5.12	3.76
Vulnerability	0.875	0.829	0.851	66.13	9.44	13.74
Total	0.893	0.856	0.874	198.51	23.78	33.5

Table 21. Measurements for the combined classifier on the Packetstorm test set

For the precision the classifier performed worse than its Packetstorm tailored counterpart in terms of product, version and postcondition categories. While product and version precision fell only marginally, the postcondition's precision fell by as much as 16.4%. The recall, however, is the same as far as the product is concerned. For the version it is slightly higher and for postcondition it went up by 17.2%. This effect is due to the larger training set which results in more postconditions being detected as the number of false negatives is reduced. Nonetheless, the amount of false positives also rises due to formulations coming from the NVD training set.

As for precondition and vulnerability categories the precision marginally rose due to the fact that the training set consisted of a much larger amount of preconditions and vulnerabilities that were annotated for the NVD.

The total results are slightly worse in terms of precision, but a bit more favourable in terms of recall. When taking a look at the F_1measure, the Packetstorm specific result is 87.8% while the F_1measure presented here yields 87.4%. These results are very close. Although the recall for the combined classifier is a little bit better than for the tailored extractor, the F_1measure for the specific classifier is slightly higher. In our case the balance between precision and recall is the main decision factor. Therefore, the F_1 measure is the preferable metric. On the one hand, information should not be accidentally added. On the other hand, as much information as possible should be detected. Moreover, product and version serve as our anti-pattern specifiers and thus, are the main point of interest in terms of Packetstorm entries. Therefore, the extractor specifically tailored to Packetstorm should be favoured.

However in terms of NVD entries, the precondition and the postcondition are of main interest, as the other data can usually be extracted in a structured manner. Product, version and vulnerability are basically additional information. Table 22 holds the results for the NVD test set.

	Precision	Recall	F_1measure	TP	FP	FN
Product	0.856	0.741	0.794	66.12	11.21	23.37
Version	0.877	0.824	0.849	72.28	10.11	15.39
Precondition	0.809	0.649	0.718	26.88	6.34	14.74
Postcondition	0.665	0.536	0.592	37.11	18.88	32.36
Vulnerability	0.737	0.600	0.661	51.77	18.59	34.45
Total	0.796	0.679	0.733	254.16	65.13	120.29

Table 22. Measurements for the combined classifier on the NVD test set

On the one hand, the precision for product and version slightly dropped as before. On the other hand, the recall went up by a bit. Furthermore, the classifier performed worse for both measures in terms of precondition. As far as the vulnerability category is considered, the precision fell while the recall slightly went up. In terms of postcondition, there is an increase of precision by 2.4% with a precision value of 66.5% with a slight negative effect on the recall. The results are generally very close to the trimmed extractor.

For the NVD the most important features are the pre- and postcondition, as the other information can mostly be extracted from the structured JSON format. Some additional information can be gained by also extracting the vulnerability which is also an influence factor for the following decision.

A look at the F_1measure reveals that the NVD extractor should be selected over the combined extractor. It is higher in both precondition and vulnerability categories and only slightly lower for the postcondition. Moreover, the overall F_1score is also preferable. Therefore, the specific extractor should be selected for this data source.

In general, the classifiers tailored to a specific type of document performed superior to the combined extractor. However, depending on the relevant information, an extractor with an overall lower scoring - but with higher scoring in a specific category - may be selected.

The results shown here could change by utilizing training sets of different sizes for Packetstorm and the NVD. Moreover, a larger training set may also increase precision and recall of all the extractors. Depending on the use case, one might use specifically tailored extractors which probably perform better on the same data source. However, if it is only possible to include one single extractor for multiple data sources, a combined ”all-purpose” classifier can be used.

6 Lessons learned

This section provides an overview on what was learned while writing this master thesis. It covers topics that came up during the planning, as well as the implementation phase.

6.1 Describing the Architecture at an Early Stage Reveals Errors

Before starting the implementation phase, the vulnerability model was planned in the form of a class diagram. Although it seemed sound in the beginning, the process of describing the classes one by one and explaining their interdependencies provided useful insight in terms of reusability and extendability of the code. What was found here is that describing a system with a first draft of the architecture can reveal planning errors and, thus help in redesigning the system. Moreover, a model should be revised multiple times before starting the implementation as it can reduce the need of time-consuming modification and redefinition in the future.

6.2 Exploitability is Preferable over Likelihood

In terms of likelihood it is an open challenge to find reliable data as it is actually a probability value. In addition, it is difficult to compute as there are many parameters involved depending on system components (to be) used. With exploitability it is feasible to act fact based. It can be determined more easily than the likelihood due to the fact that a technical description can serve as a basis for the exploitability value. Therefore, in the context of threat modelling, exploitability should be preferred over likelihood.

6.3 STRIDE is NOT for Classification

An important point that came up during the course of this master thesis is the fact that the intention behind STRIDE is to identify possible threats posed to the system or utilized components. Therefore, it is often hard to utilize it for threat classification as there may be multiple possible categories for a single threat. Consequently, it was not considered one of the main aspects, which is also why the mapping from CIA to STRIDE was chosen over a deeper additional text analysis.

6.4 A Better System Model Yields Better Results

One of the major purposes of this work is the improvement of an existing threat model. This is done by adding information concerning product specific vulnerabilities from online resources. Up-to-date information allows for fine-granular modelling with regard to real components. However, enhancing the threat model is only half the job. It is of great importance to ensure that the system model is defined as precisely as possible and contains - in the best of all cases - product specific information. This leverages the capabilities of threat modelling as not only generic information is considered during the analysis but also real-world threats and vulnerabilities are drawn into consideration. A precise threat model involves a great deal of effort for the system developer. Nevertheless, this effort can help in reducing the exploitability of the system and thus, may reduce cost in the future.

6.5 Representing Products and Versions as Rules Produces a Large Overhead

On the one hand, having a large amount of rules leverages the threat model as a digital twin. On the other hand, internal discussions have revealed that having a high amount of rules slows down the analysis process drastically, as all rules are always checked against the system model. Therefore, a more favourable approach towards this is to utilize the vulnerability model inside the analysis without the need of defining rules. This could be done by storing the data regarding the product specific vulnerabilities inside a local database. This way, the products can be be queried directly, reducing the number of rules and making the scan faster. However, this is currently not possible and would require modifications in the original application. Nonetheless, the approach will be integrated in the future.

6.6 Pre- and Postcondition Add Value

The pre- and postcondition were identified after conducting the first experiments with a very simple classifier inside the Java Stanford NER tool. The original idea was to extract product, version and vulnerability only. During the annotation process and after application of the classifier on test data, it was found that many entries contain a pre- and/or postcondition. As there were ongoing discussions on creating attack trees and attack graphs at AIT for the future, these additional categories were added to the classifiers. This enables rules that only fire if a certain precondition has been met and rules that have certain postconditions as an output. In this way, paths that an intruder may take in order to compromise the system may be revealed, which enables mitigation before a potential attack.

6.7 NER Requires Less Training if the Structure is Homogeneous

During the first experiments it was discovered that entries within Packetstorm are mostly similar in structure and wording. Thus, even a training set of 40 entries yielded promising results, which was also the reason to persue the approach of utilizing Named Entity Recognition. It can serve as a powerful tool without the need of a lot of training if the input is homogeneous enough to classify word sequences that were never annotated.

6.8 Automation Simplifies Rule Creation

Another major point discussed in this master thesis is the automation of rule creation. As the current process is very inefficient in terms of timely related factors, automation provides a speed up. By generating rules related to products based on NVD and Packetstorm, writing rules as such is not necessary. The only task required for the person validating the information is checking the input data, especially in terms of product specific information which can boost the productivity when creating rules.

6.9 There is a Vast Amount of Online Resources

The amount of resources regarding threat intelligence is on the rise. New threats are posted daily, new components are developed and new vulnerabilities are published. Furthermore, some vulnerabilities posted on various different platforms may express identical facts which requires additional analysis to omit duplicates. It is hard - if not impossible - to keep up with and to integrate all the information. However, there are efforts in joining together multiple data sources such as VulnDB. Nonetheless, platforms with different focus and different underlying data-sources are being developed and take their place in representing threat intelligence.

6.10 Outlook

In the future, additional online resources besides NVD and Packetstorm will be integrated, which will increase the coverage of known vulnerabilities. Moreover, classifiers will be trained on larger training sets in order to enhance the quality of the entity recognition task. In addition, extracted pre- and postconditions will serve as input for automatic attack graph generation to help experts understand potential attack chains of individual attack steps (Lallie et al., 2020; Phillips & Swiler, 1998; Ammann, Pamula, Street, & Ritchey, 2005) by visualizing cyber attacks. It has to be considered that risk depends on the actual situation and that a risk matrix containing crisp borders may not be applicable in all application domains. Therefore, a fuzzy system allowing for more flexible classification could be utilized in the future. All these aforementioned points will provide further insight into the threat landscape and, thus, lead to an improved risk management process. An abstraction and generalization of the results achieved in this thesis might in an ideal case lead to their application in the wider field of cyber threat intelligence.

7 Conclusion

Threat modelling serves as a means to reveal posed threats (Torr, 2005) as well as vulnerabilities that a system under consideration suffers from. It is an iterative process (Ma & Schmittner, 2016) that compares a digital twin of a real-world system with a digital twin of threats and vulnerabilities, and checks for patterns that should not be contained within the system model. We illustrated the current process underlying threat modelling on the basis of an example and performed an exemplary risk management process. However, until then we were working with focus on engineering phases as well as generic components and did not consider the exact products. As every component comes with its own respective vulnerabilities this is a definite shortcoming. Consequently, we searched for an approach to extend the underlying threat model with information on commonly known vulnerabilities and found out that vast amounts of resources are shared online.

Information sharing enables faster incident response towards the newest threats and provides the possibility to reveal and, consequently, act on vulnerabilities (Tounsi & Rais, 2017; ENISA, 2018). Information comes in various formats and is shared via distinct platforms, each with a different focus. All these platforms have their own strengths and weaknesses (Wang & Chow, 2019; Li et al., 2019). Some are even focused on specific domains. Utilizing intelligence from multiple sources facilitates a more sophisticated analysis.

We focused on vulnerability intelligence and inherent vulnerability databases as they contain long-term valid information relevant to threat modelling. Moreover, the different formats and measures provided by these databases all take their rightful place in threat and vulnerability management. Taking these formats and measures into account, a custom format capable of integrating various data sources was derived. By gathering descriptions, scores, references, associated weaknesses and affected products it becomes feasible to pinpoint vulnerabilities.

In addition, possible mappings for CIA to STRIDE and CVSS to an ISO/SAE DIS 21434 based risk matrix were elaborated on. While the CIA to STRIDE mapping did not produce outstanding results as STRIDE is basically an extended inversion of CIA, the CVSS mapping was more promising. This is due to the fact that it is feasible to create a custom mapping without major restrictions. The CVSS base score, its exploitability and the impact were mapped to a 4x4 matrix with severity levels of 1 to 5. This lead to a more critical, asymmetric risk matrix (table 14) than the default symmetric one (table 1). The resulting risk matrix is relevant for risk management and the respective risk treatment options to be chosen. However, the discussed matrix includes crisp borders, which in some cases results in an inaccurate classification of the severity level.

Although many vulnerability databases offer information in a structured format, not all information can be retrieved in this way. Parts of this information, especially descriptions, are formulated in plain text and thus require further analysis as they cannot be directly ingested by a machine (Syed et al., 2016; Wang & Chow, 2019; Ramnani et al., 2017) and hereupon be utilized for threat analysis. In order to also allow for this type of data to be included into the threat model, an information extraction task was applied. Information extraction, which in this case is Named Entity Recognition, enables the analysis of text sequences and their respective categorization (Jurafsky & Martin, 2000; Bird et al., 2009). In this way it becomes feasible to put word sequences into distinct categories and utilize the extracted information in future threat analyses.

To join all the retrieved information together, a prototype has been developed. It is capable of extracting information from arbitrary sources utilizing the adapter pattern. Currently, one adapter for Packetstorm and another one for CVE/NVD have been implemented. Moreover, the prototype displays the retrieved data in order for the end user to validate and, eventually, adapt the presented content in case the NER classifier has not correctly identified all relevant entities. Furthermore, the prototype handles the discussed mappings by outputting data according to the custom vulnerability model and stores them in a database. This enables automated rule generation from the stored vulnerability information even for arbitrary grammars. For now, only vulnerabilities contained in Packetstorm and the CVE/NVD have been integrated. However, in the future this approach could be adapted and extended to process more complex data sources.

Towards the end of this thesis, we evaluated test results for three entity extractors, of which all provided promising results. It was found out that the classifiers that were trained only on one specific resource produced superior results compared to an entity extractor based on multiple distinct datasets. This is due to the fact that homogeneous structure bears an advantage when training an NER system. The greater the similarity between training and test data, the bigger precision, recall and their respective F_1measure will be. This is also why the CVE/NVD results are inferior compared to the results from Packetstorm. CVE entries are submitted by public sources and then sent to CVE Numbering Authorities of the respective state before being published. This is why vulnerabilities often come in different formulations, which leads to more heterogeneity in the dataset. In general, the datasets were rather small. A larger dataset could improve the resulting extractors. This is particularly relevant for heterogeneous data sources encompassing many outliers which could affect the results negatively.

Before this work, it was not feasible to create product specific rules efficiently. This is due to the fact that the amount of data presented online can hardly be managed manually by experts. It becomes apparent when considering that the CVE alone contains more than 140.000 vulnerabilities. Although it is the de-facto standard for vulnerability enumeration, it does not cover all existing vulnerabilities, which is why further databases have been developed and even more are expected to come. By automating rule creation, it becomes possible to establish rules in a standardized way, meaning that all of the resulting rules are generated utilizing the same structure as their basis.

New vulnerabilities are detected day by day. As security experts it is our duty to prevent possible attacks as best as possible. Therefore, it is essential to pay attention to the ever-evolving risk landscape in order to make our systems more resilient and to inform decisions based on detected threats and vulnerabilities. Staying up-to-date with current developments and including them into all phases of the product lifecycle, both in terms of an updated system - as well as threat model, will reduce the number of cyber-security vulnerabilities and their inherent threats. Automation in terms of rule generation and data extraction from real-world information can serve as an important tool in reaching this goal and improve the current threat model.

References

0day Today Team. (2020). *0day.today Exploit Database.* Retrieved 2020-08-31, from https://0day.today/

Abomhara, M., Køien, G., & Gerdes, M. (2015). *A STRIDE-Based Threat Model for Telehealth Systems.*

AlienVault. (2020, April). Retrieved 2020-04-29, from https://otx.alienvault.com/ (Library Catalog: otx.alienvault.com)

Ammann, P., Pamula, J., Street, J., & Ritchey, R. (2005). A Host-Based Approach to Network Attack Chaining Analysis. In *21st Annual Computer Security Applications Conference (ACSAC'05)* (pp. 72–84). Tucson, AZ, USA: IEEE. Retrieved 2020-05-07, from http://ieeexplore.ieee.org/document/1565236/ doi: https://doi.org/10.1109/CSAC.2005.6

Anomali. (2020). *ThreatStream - Threat Intelligence Platform.* Retrieved 2020-05-04, from https://www.anomali.com/products/threatstream (Library Catalog: www.anomali.com)

Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python* (1st ed.). O'Reilly Media, Inc.

Byers, B., & Owen, H. (2019, September). Automation Support for CVE Retrieval. , 21.

Che, W., Wang, M., Manning, C. D., & Liu, T. (2013). Named Entity Recognition with Bilingual Constraints. , 11.

Cheikes, B., Waltermire, D., & Scarfone, K. (2011, August). *Common Platform Enumeration: Naming Specification Version 2.3* (Tech. Rep. No. NIST Internal or Interagency Report (NISTIR) 7695). National Institute of Standards and Technology. Retrieved 2020-08-05, from https://csrc.nist.gov/publications/detail/nistir/7695/final doi: https://doi.org/https://doi.org/10.6028/NIST.IR.7695

Chismon, D., & Ruks, M. (2015). *Threat Intelligence: Collecting, Analysing, Evaluating.* MWR Infosecurity.

The Cyber Vault Project. (2020, April). Retrieved 2020-04-20, from https://nsarchive.gwu.edu/project/cyber-vault-project

DBpedia. (2019). Retrieved 2020-05-03, from https://wiki.dbpedia.org/

Desmet, L., Jacobs, B., Piessens, F., & Joosen, W. (2005). Threat Modelling for Web Services Based Web Applications. In D. Chadwick & B. Preneel (Eds.), *Communications and Multimedia Security* (pp. 131–144). Boston, MA: Springer US.

Eiram, C. (2018, January). *What You Don't Know About The Vulnerability Ecosystem Can Lead To A Data Breach.* Retrieved 2020-10-05, from https://www.riskbasedsecurity.com/2018/01/17/what-you-dont-know-about-the-vulnerability-ecosystem-can-lead-to-a-data-breach/ (Section: News)

ENISA. (2014a). *Actionable information for security incident response.* Author. Retrieved 2020-04-17, from https://www.enisa.europa.eu/publications/actionable-information-for-security

ENISA. (2014b). *Standards and tools for exchange and processing of actionable information.* Author. Retrieved 2020-04-17, from https://www.enisa.europa.eu/publications/standards-and-tools-for-exchange-and-processing-of-actionable-information

ENISA. (2017). *Exploring the opportunities and limitations of current Threat Intelligence Platforms.* Author. Retrieved 2020-04-17, from https://www.enisa.europa.eu/publications/exploring-the-opportunities-and-limitations-of-current-threat-intelligence-platforms

ENISA. (2018). *ENISA Threat Landscape Report 2018.* Author. Retrieved 2020-04-17, from https://www.enisa.europa.eu/publications/enisa-threat-landscape-report-2018

ENISA. (2020a). *The Risk Management Process* [Page]. Retrieved 2020-11-26, from https://www.enisa.europa.eu/topics/threat-risk-management/risk-management/current-risk/risk-management-inventory/rm-process/rm-process

ENISA. (2020b). *Risk Treatment* [Page]. Retrieved 2020-12-01, from https://www.enisa.europa.eu/topics/threat-risk-management/risk-management/current-risk/risk-management-inventory/rm-process/risk-treatment/risk-treatment

Facebook. (2020). *Facebook ThreatExchange.* Retrieved 2020-05-04, from https://developers.facebook.com/programs/threatexchange/ (Library Catalog: developers.facebook.com)

Finkel, J. R., Grenager, T., & Manning, C. (2005). Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics - ACL '05* (pp. 363–370). Ann Arbor, Michigan: Association for Computational Linguistics. Retrieved 2020-10-09, from http://portal.acm.org/citation.cfm?doid=1219840.1219885 doi: https://doi.org/10.3115/1219840.1219885

FIRST. (2020a). *Common Vulnerability Scoring System SIG.* Retrieved 2020-05-04, from https://www.first.org/cvss (Library Catalog: www.first.org)

FIRST. (2020b). *Common Vulnerability Scoring System version 3.1 Specification Document Revision 1.* Retrieved 2020-08-05, from https://www.first.org/cvss/v3-1/cvss-v31-specification_r1.pdf

Freeman, E., Robson, E., Bates, B., & Sierra, K. (2014). *Head first design patterns* (Second release ed.). Beijing Boston Famham Sebastopol Tokyo: O'Reilly. (OCLC: 1085988254)

Goyal, A., Kumar, M., & Gupta, V. (2017, October). Named Entity Recognition: Applications, Approaches and Challenges. , 15.

Granova, A., & Slaviero, M. (2014). Cyber Warfare. In *Cyber Security and IT Infrastructure Protection* (pp. 205–232). Elsevier. Retrieved 2020-10-05, from https://linkinghub.elsevier.com/retrieve/pii/B9780124166813000082 doi: https://doi.org/10.1016/B978-0-12-416681-3.00008-2

Hamad, M. (2020). *A Multilayer Secure Framework for Vehicular Systems.* Retrieved 2020-08-28, from https://www.researchgate.net/publication/341597533_A_Multilayer_Secure_Framework_for_Vehicular_Systems

Honnibal, M., & Montani, I. (2017). *spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.* (To appear)

Horev, R. (2018, November). *BERT Explained: State of the art language model for NLP.* Retrieved 2020-05-04, from https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270 (Library Catalog: towardsdatascience.com)

Hussain, S., Kamal, A., Rasool, G., & Iqbal, S. (2014, January). Threat Modelling Methodologies: A Survey. , *26*, 1607–1609.

IBM. (2020). *IBM X-Force Exchange.* Retrieved 2020-05-04, from https://exchange.xforce.ibmcloud.com/ (Library Catalog: exchange.xforce.ibmcloud.com)

IETF. (2020, May). *IETF Homepage.* Retrieved 2020-05-04, from https://www.ietf.org/ (Library Catalog: www.ietf.org)

ISO/TC 22/SC 32. (2020). *ISO/SAE DIS 21434 Road vehicles — Cybersecurity engineering.* ISO - International Standardization Organization. Retrieved 2020-05-01, from https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/07/09/70918.html

ISO/TC 262 Risk management. (2018). *ISO 31000 - Risk management - guidelines.* (No. 31000). (OCLC: 1030875208)

Jaeger, D., Ussath, M., Cheng, F., & Meinel, C. (2015, November). Multistep Attack Pattern Detection on Normalized Event Logs. In *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing* (pp. 390–398). New York, NY, USA: IEEE. Retrieved 2020-04-24, from http://ieeexplore.ieee.org/document/7371512/ doi: https://doi.org/10.1109/CSCloud.2015.26

Jha, S., Sheyner, O., & Wing, J. (2002). Two formal analyses of attack graphs. In *Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15* (pp. 49–63). Cape Breton, NS, Canada: IEEE Comput. Soc. Retrieved 2020-04-24, from http://ieeexplore.ieee.org/document/1021806/ doi: https://doi.org/10.1109/CSFW.2002.1021806

Joshi, A., Lal, R., Finin, T., & Joshi, A. (2013, September). Extracting Cybersecurity Related Linked Data from Text. In *2013 IEEE Seventh International Conference on Semantic Computing* (pp. 252–259). (ISSN: null) doi: https://doi.org/10.1109/ICSC.2013.50

Jurafsky, D., & Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* (1st ed.). USA: Prentice Hall PTR.

Kaspersky. (2020). *Threat Intelligence Definition. Why Threat Intelligence Is Important for Your Business and How to Evaluate a Threat Intelligence Program.* Retrieved 2020-04-26, from https://www.kaspersky.com/

resource-center/definitions/threat-intelligence (AO Kaspersky Lab)

Kouraklis, J. (2019, August). In the Land of ORM. In (pp. 1–18).

Lallie, H. S., Debattista, K., & Bal, J. (2020, February). A review of attack graph and attack tree visual syntax in cyber security. *Computer Science Review*, *35*, 100219. Retrieved 2020-04-25, from https://linkinghub.elsevier.com/retrieve/pii/S1574013719300772 doi: https://doi.org/10.1016/j.cosrev.2019.100219

Laorden, C., Sanz, B., Alvarez, G., & Bringas, P. (2010, January). A Threat Model Approach to Threats and Vulnerabilities in On-line Social Networks. In (Vol. 85, pp. 135–142).

Lautenbach, A., & Islam, M. (2016, March). *HEAling Vulnerabilities to ENhance Software Security and Safety - Security models.* Retrieved 2020-08-28, from https://autosec.se/wp-content/uploads/2018/03/HEAVENS_D2_v2.0.pdf

Lei, J., Tang, B., Lu, X., Gao, K., Jiang, M., & Xu, H. (2014, September). A comprehensive study of named entity recognition in Chinese clinical text. *Journal of the American Medical Informatics Association*, *21*(5), 808–814. Retrieved 2020-08-11, from https://academic.oup.com/jamia/article/21/5/808/758234 (Publisher: Oxford Academic) doi: https://doi.org/10.1136/amiajnl-2013-002381

Li, V. G., Dunn, M., Pearce, P., McCoy, D., Voelker, G. M., Savage, S., & Levchenko, K. (2019). Reading the Tea Leaves: A Comparative Analysis of Threat Intelligence. , 18.

LookingGlass. (2020). *scoutTHREAT Threat Intelligence Platform | LookingGlass.* Retrieved 2020-05-04, from https://www.lookingglasscyber.com/products/threat-platforms/scoutthreat/ (Library Catalog: www.lookingglasscyber.com)

Ma, Z., & Schmittner, C. (2016, November). Threat Modeling for Automotive Security Analysis. In (pp. 333–339). doi: https://doi.org/10.14257/astl.2016.139.68

Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., & McClosky, D. (2014). The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations* (pp. 55–60). Baltimore, Maryland: Association for Computational Linguistics. Retrieved 2020-10-25, from http://aclweb.org/anthology/P14-5010 doi: https://doi.org/10.3115/v1/P14-5010

Mariani, S. (2020, September). *Phat3/CVE-analyzer.* Retrieved 2020-11-26, from https://github.com/Phat3/CVE-analyzer (original-date: 2018-10-29T18:19:27Z)

Mehler-Bicher, A., Mehler, F., Kuntze, N., Kunz, S., Ostheimer, B., Steiger, L., & Weih, H.-P. (2019). *Wirtschaftsinformatik Klipp und Klar.* Wiesbaden: Springer Fachmedien Wiesbaden. Retrieved 2020-10-25, from http://link.springer.com/10.1007/978-3-658-26494-9 doi:

https://doi.org/10.1007/978-3-658-26494-9

Meland, P. H., Paja, E., Gjære, E. A., Paul, S., Dalpiaz, F., & Giorgini, P. (2014, April). Threat Analysis in Goal-Oriented Security Requirements Modelling. *International Journal of Secure Software Engineering*, *5*, 1–19. doi: https://doi.org/10.4018/ijsse.2014040101

Menges, F., & Pernul, G. (2018, March). A comparative analysis of incident reporting formats. *Computers & Security*, *73*, 87–101. Retrieved 2020-04-14, from http://www.sciencedirect.com/science/article/pii/S0167404817302250 doi: https://doi.org/10.1016/j.cose.2017.10.009

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013, September). Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781 [cs]*. Retrieved 2020-04-26, from http://arxiv.org/abs/1301.3781 (arXiv: 1301.3781)

Miller, S. (2018, January). *What is Strategic Threat Intelligence?* Retrieved 2020-05-03, from https://www.anomali.com/blog/what-is-strategic-threat-intelligence (Library Catalog: www.anomali.com)

MISP. (2020, April). Retrieved 2020-04-29, from https://www.misp-project.org/

MITRE. (2019a, August). *CVE and NVD Relationship.* Retrieved 2020-04-17, from https://cve.mitre.org/about/cve_and_nvd_relationship.html

MITRE. (2019b, November). *CVE - Home.* Retrieved 2020-08-05, from https://cve.mitre.org/about/index.html

MITRE. (2020a, April). *CVE - Common Vulnerabilities and Exposures (CVE).* Retrieved 2020-04-29, from https://cve.mitre.org/

MITRE. (2020b, July). *CVE - CVE Numbering Authorities.* Retrieved 2020-08-05, from https://cve.mitre.org/cve/cna.html#submitting_cve_entry_info

MITRE. (2020c, February). *CWE - About - CWE Overview.* Retrieved 2020-08-05, from https://cwe.mitre.org/about/index.html

MITRE. (2020d, June). *CWE - CWE-121: Stack-based Buffer Overflow (4.2).* Retrieved 2020-11-25, from https://cwe.mitre.org/data/definitions/121.html

MITRE. (2020e, April). *The MITRE Corporation.* Retrieved 2020-04-20, from https://www.mitre.org/

Mittal, S., Das, P. K., Mulwad, V., Joshi, A., & Finin, T. (2016, August). CyberTwitter: Using Twitter to generate alerts for cybersecurity threats and vulnerabilities. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)* (pp. 860–867). doi: https://doi.org/10.1109/ASONAM.2016.7752338

Mittal, S., Joshi, A., & Finin, T. (2019, May). Cyber-All-Intel: An AI for Security related Threat Intelligence. *arXiv:1905.02895 [cs]*. Retrieved 2020-01-10, from http://arxiv.org/abs/1905.02895 (arXiv: 1905.02895)

Mulwad, V., Li, W., Joshi, A., Finin, T., & Viswanathan, K. (2011, August). Extracting Information about Security Vulnerabilities from Web Text. In (Vol. 3, pp. 257–260). doi: https://doi.org/10.1109/WI-IAT.2011.26

Nakayama, H., Kubo, T., Kamura, J., Taniguchi, Y., & Liang, X. (2018). *doccano: Text annotation tool for human.* Retrieved from https://github.com/doccano/doccano (Software available from https://github.com/doccano/doccano)

NIS Directive. (2016, July). Directive (EU) 2016/1148 of the European Parliament and of the Council of 6 July 2016 concerning measures for a high common level of security of network and information systems across the Union. (194). Retrieved 2020-05-04, from http://data.europa.eu/eli/dir/2016/1148/oj/eng (Code Number: 194)

NIST. (2020a, April). *National Institute of Standards and Technology* [text]. Retrieved 2020-04-20, from https://www.nist.gov/ (Library Catalog: www.nist.gov)

NIST. (2020b, April). *NVD.* Retrieved 2020-04-29, from https://nvd.nist.gov/

NIST. (2020c). *NVD CWE Slice.* Retrieved 2020-08-05, from https://nvd.nist.gov/vuln/categories

NIST. (2020d). *NVD - General Information.* Retrieved 2020-10-05, from https://nvd.nist.gov/general

OASIS CTI TC. (2020, April). Retrieved 2020-04-20, from https://oasis-open.github.io/cti-documentation/

Ou, X., Boyer, W. F., & McQueen, M. A. (2006). A scalable approach to attack graph generation. In *Proceedings of the 13th ACM conference on Computer and communications security - CCS '06* (pp. 336–345). Alexandria, Virginia, USA: ACM Press. Retrieved 2020-04-24, from http://dl.acm.org/citation.cfm?doid=1180405.1180446 doi: https://doi.org/10.1145/1180405.1180446

Packetstorm. (2020). *About Packet Storm.* Retrieved 2020-10-05, from https://packetstormsecurity.com/about/

Phillips, C., & Swiler, L. P. (1998). A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms - NSPW '98* (pp. 71–79). Charlottesville, Virginia, United States: ACM Press. Retrieved 2020-04-24, from http://portal.acm.org/citation.cfm?doid=310889.310919 doi: https://doi.org/10.1145/310889.310919

Pingle, A., Piplai, A., Mittal, S., & Joshi, A. (2019). *RelExt: Relation Extraction using Deep Learning approaches for Cybersecurity Knowledge Graph Improvement.*

Pinkston, J., Undercoffer, J., Joshi, A., & Finin, T. (2003, June). A Target-Centric Ontology for Intrusion Detection.

Ponemon. (2016). *The Value of Threat Intelligence: A Study of North American & United Kingdom Companies.* Ponemon Institute.

Ponemon. (2019). *The Value of Threat Intelligence: Annual Study of North American & United Kingdom Companies.* Ponemon Institute. Retrieved from https://stratejm.com/wp-content/uploads/2019/08/2019_Ponemon_Institute-Value_of_Threat_Intelligence_Research

_Report_from_Anomali.pdf

Ramnani, R., Shivaram, K., Sengupta, S., & M., A. (2017, February). Semi-Automated Information Extraction from Unstructured Threat Advisories. In (pp. 181–187). doi: https://doi.org/10.1145/3021460.3021482

Ross, R., Pillitteri, V., Dempsey, K., Riddle, M., & Guissanie, G. (2020, February). *Protecting controlled unclassified information in nonfederal systems and organizations* (Tech. Rep. No. NIST SP 800-171r2). Gaithersburg, MD: National Institute of Standards and Technology. Retrieved 2020-11-10, from https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-171r2.pdf doi: https://doi.org/10.6028/NIST.SP.800-171r2

Satyapanich, T. W., Finin, T., & Ferraro, F. (2019, December). Extracting Rich Semantic Information about Cybersecurity Events. *Second Workshop on Big Data for CyberSecurity, held in conjunction with the IEEE Int. Conf. on Big Data.* Retrieved 2020-01-13, from https://ebiquity.umbc.edu/paper/html/id/873/Extracting-Rich-Semantic-Information-about-Cybersecurity-Events

Sauerwein, C., Sillaber, C., Mussmann, A., & Breu, R. (2017). Threat Intelligence Sharing Platforms: An Exploratory Study of Software Vendors and Research Perspectives. In *Wirtschaftsinformatik.*

Schmittner, C., Tummeltshammer, P., Hofbauer, D., Shaaban, A., Meidlinger, M., Tauber, M., … Brandstetter, M. (2019, January). Threat Modeling in the Railway Domain. In (pp. 261–271).

Shevchenko, N. (2018, December). *Threat Modeling: 12 Available Methods.* Retrieved 2020-10-05, from https://insights.sei.cmu.edu/sei_blog/2018/12/threat-modeling-12-available-methods.html

Shostack, A. (2014). *Threat modeling: designing for security.* Indianapolis, IN: Wiley. (OCLC: 855043351)

Sillaber, C., Sauerwein, C., Mussmann, A., & Breu, R. (2016, October). Data Quality Challenges and Future Research Directions in Threat Intelligence Sharing Practice. In *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security* (pp. 65–70). Vienna, Austria: Association for Computing Machinery. Retrieved 2020-04-19, from https://doi.org/10.1145/2994539.2994546 doi: https://doi.org/10.1145/2994539.2994546

Strobl, S., Hofbauer, D., Schmittner, C., Maksuti, S., Tauber, M., & Delsing, J. (2018, May). Connected cars — Threats, vulnerabilities and their impact. In *2018 IEEE Industrial Cyber-Physical Systems (ICPS)* (pp. 375–380). doi: https://doi.org/10.1109/ICPHYS.2018.8387687

stucco-archive/extractors. (2019, August). stucco-archive. Retrieved 2020-04-26, from https://github.com/stucco-archive/extractors (original-date: 2013-07-30T18:57:48Z)

stucco/entity-extractor. (2020, March). Stucco. Retrieved 2020-04-21, from https://github.com/stucco/entity-extractor (original-date: 2014-01-05T13:23:45Z)

Swanson, M., Bowen, P., Phillips, A. W., Gallup, D., & Lynes, D. (2010). *Contingency planning guide for federal information systems* (Tech. Rep. No. NIST SP 800-34r1). Gaithersburg, MD: National Institute of Standards and Technology. Retrieved 2020-11-10, from `https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-34r1.pdf` (Edition: 0) doi: https://doi.org/10.6028/NIST.SP.800-34r1

Syed, Z., Padia, A., Finin, T., Mathews, L., & Joshi, A. (2016, February). UCO: A Unified Cybersecurity Ontology..

Thelen, M., & Riloff, E. (2002). A bootstrapping method for learning semantic lexicons using extraction pattern contexts. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing - EMNLP '02* (Vol. 10, pp. 214–221). Not Known: Association for Computational Linguistics. Retrieved 2020-05-03, from `http://portal.acm.org/citation.cfm?doid=1118693.1118721` doi: https://doi.org/10.3115/1118693.1118721

theresilience. (2018, June). *Risk Treatment Methods.* Retrieved 2020-10-03, from `https://www.theresilience.ml/risk-treatment-methods/`

ThreatConnect. (2020). *ThreatConect Homepage.* Retrieved 2020-05-04, from `https://threatconnect.com/` (Library Catalog: threatconnect.com)

Torr, P. (2005, September). Demystifying the Threat-Modeling Process. *IEEE Security and Privacy Magazine*, *3*(5), 66–70. Retrieved 2020-07-29, from `http://ieeexplore.ieee.org/document/1514406/` doi: https://doi.org/10.1109/MSP.2005.119

Tounsi, W., & Rais, H. (2017, September). A survey on technical threat intelligence in the age of sophisticated cyber attacks. *Computers & Security*, *72*. doi: https://doi.org/10.1016/j.cose.2017.09.001

UNECE. (2020, June). *UN Regulations on Cybersecurity and Software Updates to pave the way for mass roll out of connected vehicles.* Retrieved 2020-10-09, from `http://www.unece.org/?id=54667`

Vadapalli, S. R., Hsieh, G., & Nauer, K. S. (2018). *TwitterOSINT: Automated Cybersecurity Threat Intelligence Collection and Analysis using Twitter Data.* Place of publication not identified: CSREA Press.

The VERIS Framework. (2020). Retrieved 2020-04-20, from `http://veriscommunity.net/`

Wagner, C., Dulaunoy, A., Wagener, G., & Iklody, A. (2016, October). MISP: The Design and Implementation of a Collaborative Threat Intelligence Sharing Platform. In *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security* (pp. 49–56). Vienna, Austria: Association for Computing Machinery. Retrieved 2020-04-12, from `https://doi.org/10.1145/2994539.2994542` doi: https://doi.org/10.1145/2994539.2994542

Wang, T., & Chow, K. P. (2019, July). Automatic Tagging of Cyber Threat Intelligence Unstructured Data using Semantics Extraction. In *2019 IEEE International Conference on Intelligence and Security Informatics (ISI)* (pp. 197–199). doi: https://doi.org/10.1109/ISI.2019.8823252

Zheng, D. E., & Lewis, J. A. (2015). Cyber Threat Information Sharing: Recommendations for Congress and the Administration. , 18.

8 Annex A: Definition of Terms

Availability describes the timely and reliable access of information and services. (Swanson, Bowen, Phillips, Gallup, & Lynes, 2010; Ross, Pillitteri, Dempsey, Riddle, & Guissanie, 2020)

Confidentiality deals with access restrictions and, consequently, allowing only authorized staff to access information. Moreover, confidentiality includes means for the protection of privacy and prevents information disclosure. (Swanson et al., 2010; Ross et al., 2020)

Exploits abuse vulnerabilities that are located within the target system. They make use of preconditions in order to compromise the system and possibly cause damage. Furthermore, a successful exploit can result in new preconditions that allow for deeper access into the system and, consequently, further exploitation. (Lallie et al., 2020)

Exploitability is an alternative measure of likelihood. It does not depend on a probability value but rather utilizes categories implying the exposure of the system. To be more precise, exploitability describes the attack surface and, consequently, the effort and the required knowledge necessary for launching an attack (FIRST, 2020b). It can be based on a technical description of the system under consideration and the experience of experts.

Impact describes the magnitude of the damage that results from a potential exploit. It deals with disclosure (confidentiality impact), modification (integrity impact), and loss of information (availability impact) which are subject to unauthorized influence (Swanson et al., 2010). Impact describes the harmful effect on an organization's operations, assets, individuals or external entities that an exploit may result in (Ross et al., 2020). Moreover, it characterizes the impact on safety, financial, operational and privacy aspects (ISO/TC 22/SC 32, 2020).

Integrity depicts protection against unauthorized modification or destruction of information (Swanson et al., 2010; Ross et al., 2020). It also includes non-repudiation and authenticity. Therefore, it can be referred to as granting the veracity and trustworthiness of information (FIRST, 2020b).

Likelihood is a value describing the probability of an exploit or an occurring risk. It generally depends on a lot of parameters, particularly in the cybersecurity sector, which is why this thesis replaces it with the term "exploitability".

Precondition describes system properties that must be given in order to conduct a successful exploit or attack step. For example, a precondition manifests due to the reachability from the outside of a component or service. Moreover, a certain software known to be vulnerable only on a specific operating system may form a precondition. Also convincing a user to install malicious software could be a precondition for a successful exploit. For a vulnerability to be exploitable there must always be a precondition. All further attack steps could be prevented by addressing the initial precondition (Lallie et al., 2020). However, a precondition could lead to a postcondition, which may then be utilized as the next precondition. E.g., a postcondition which discloses information could allow an attacker to retrieve an admin password which could then be used as precondition for additional attack steps.

Postcondition describes the result of an exploit and capabilities that the attacker gains through a successful attack. One exploit may have one or multiple postconditions. Moreover, a postcondition may become a precondition for further exploits or attack steps. (Lallie et al., 2020)

Rules are anti-patterns that the system model is checked against in order to identify threats and vulnerabilities. They represent potentially vulnerable system components or component configurations known to be insecure in terms of both software and hardware. Moreover, they support the risk management process by holding name, description, impact and exploitability values, links to advisories, as well as pre- and postconditions.

Risk represents a measure of how much a system (component) is threatened by certain circumstances or situations. In our case, risk is portrayed by the severity level represented in the risk matrix. It is a function of the impact and the exploitability of a certain event. (Ross et al., 2020)

Threat depicts any situation or circumstance leading to a potentially (negative) impact on confidentiality, integrity, availability, organizations, assets, as well as external entities. (Ross et al., 2020)

Vulnerability describes an exploitable weakness in the system under consideration. Moreover, a vulnerability includes the affected products as well as the exact versions that suffer from this weakness. It results from flaws in the design, implementation or management of the system. Vulnerabilities emerge due to preconditions that allow for an exploit within the system under consideration. (Lallie et al., 2020)

Weaknesses express flaws, faults or bugs in either software or hardware. These weaknesses may result in a vulnerable system and may arise due to planning, coding or architectural errors. (MITRE, 2020c)

9 Annex B: Abbreviations

CAPEC	Common Attack Pattern Enumeration and Classification
CIA	Confidentiality, Integrity, Availability
CPE	Common Platform Enumeration
CRF	Conditional Random Fields
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
CVSS	Common Vulnerabilities Scoring System
DIS	Draft International Standard
ENISA	European Union Agency for Cybersecurity (originally: European Network and Information Security Agency)
GUI	Graphical User Interface
ISO	International Standardization Organization
MIME	Multipurpose Internet Mail Extensions
NER	Named Entity Recognition
NIST	National Institute of Standards and Technology
NLP	Natural Language Processing
NVD	National Vulnerability Database
OWL	Web Ontology Language
RDF	Resource Description Format
SAE	Society of Automotive Engineers
SQL	Structured Query Language
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege
UN	United Nations
UNECE	United Nations Economic Commission for Europe

10 Annex C: Implementation

The implementation, as well as the gradle build script are located on the DVD. In case you want to take a look at the implementation in detail and reproduce the measurements, the whole source code is located on the DVD and can be imported into an IDE by importing the project folder called "Rule_Extractor" as gradle project (it has been tested with eclipse). If you want to try out the prototype please follow the installation guide which is provided in the following section.

10.1 Installation Guide

In order to create an executable version of the prototype, Java 8 JDK is required. The supported database is PostgreSQL, which is required for the prototype to store the data. The DVD contains the following resources:

- **jdk-8u161-windows-x64.exe** - an installer for the Java 8 jdk
- **postgresql-10.15-1-windows-x64.exe** - an installer for the postgresql database in version 10
- **Rule_Extractor** - the gradle project containing the source code as well as the resources
    * **src/main/java/at/ac/ait** contains all the Java code relevant for the application
        - **adapter** contains all the adapter interface, its implementations, a list of all utilized file names throughout the application as well as a wrapper for named classified entities
        - **annotated** contains classes generated from the schema utilized in doccano json files
        - **jsonclassgen** contains a class able to generate Java classes from a given json file. It includes its own "main" method and can only be used during development
        - **main/model** contains all classes of the vulnerability model
        - **main/model/enumeration** contains all the enums relevant to store the type of product, the cvss value, the type of entity, as well as value for impact, exploitability and STRIDE
        - **NVD/types** contains all classes generated from the NVD json file. It represents an intermediate format which is later mapped to the vulnerability model
        - **rulegen** contains the "RuleGenerator" class which is the entrypoint of the application It starts the GUI but also allows for importing datasets to do training and measurements. However, this is only possible in development mode by changing the code. Moreover, the folder contains the GUIController as well as intermediate classes required to represent and modify data within the GUI
    * **src/main/resources/at/ac/ait/files** contains all the resources utilized within the application (changes can only be applied in development mode e.g. within an IDE)

  - **annotated** contains the datasets created with doccano ending with ”.json1”. Training and test sets that are automatically separated and created from these ”.json1” are also located here. It also contains the ”gzipped” entity extractors that result from training
  - **gui** contains an ”fxml” representation of the GUI
  - **NVD** contains a demonstration file regarding NVD data utilized in the final application
  - **PS** contains a demonstration file regarding Packetstorm data utilized in the final application
  - **META-INF** contains a ”logback.xml” file to prohibit output from Hibernate as this would otherwise slow down the application. ”persistence.xml” contains the configuration for the database connection and script creation
- **Rule_Extractor.jar** - the executable jar file
- **ruleext.dump** - the dump of the database, containing the schema

Please make sure to install the Java JDK and PostgreSQL for the application to work. The JDK as well as the PostgreSQL can be installed with default configuration. Make sure to also install pgAdmin which is also contained in the PostgreSQL installation. As password for PostgreSQL choose ”root” (without the quotation marks). This is preconfigured for the application. However, in case you have an existing PostgreSQL database in version 10, it is possible to change the ”value” attributes in following lines of code in ”META-INF/persistence.xml” to adapt password, username and the url to the database.

```
<property name=”javax.persistence.jdbc.url”
value=”jdbc:postgresql://localhost:5432/ruleext” />

<property name=”javax.persistence.jdbc.user” value=”postgres” />

<property name=”javax.persistence.jdbc.password” value=”root” />
```

Once everything has been installed, open pgAdmin. It should be possible to search for it in the searchbar. When asked for the password type in ”root” (without the quotation marks). Now expand the ”Servers” and the ”PostgreSQL 10” items in the browser.

Right click ”Databases” an navigate to ”Create > Database...”. This will open a new window. Set the field called ”Database” to ”ruleext” and ensure that ”UTF8” is specified as ”Encoding” in the ”Definition” Tab. Click the ”Save” button. This will create the ”ruleext” database.

Now click on the text ”ruleext” (although the whole line is marked it does not select it otherwise) and right-click it afterwards. This will open a context menu. Inside this context menu select ”Restore...”. Again this will open a window. The only thing that needs to be done here is to click on the three dots next to the file name and select the ”ruleext.dump” located on the DVD. If the file is not displayed in the provided explorer make sure to specify the Format (in the right bottom corner of the window) as "All Files". Once "ruleext.dump" has been selected, click on "Select" in the right bottom corner. Finally, clicking on "Restore" will create a database from the provided schema.

The DVD comes with an executable ".jar" file. In case you need to change the configuration or want to build the application yourself, it is possible to execute the "build.bat" which executes a gradle build and generates a ".jar" file. A build can also be triggered manually by navigating to the "Rule_Extractor" folder in the command prompt and typing "gradlew jar". When utilizing the "build.bat" the resulting ".jar" file can be found in the same folder as the "build.bat" is located. When building manually, it will be located in "Rule_Extractor\build \libs".

Double clicking on the resulting ".jar" file will open the application.

10.2 User Guide

The components of the GUI have already been explained in section 5.1.5. This section will provide information on how the user can interact with the GUI.

10.2.1 Selecting the Resource for Rule Extraction

The GUI provides two buttons in the upper left corner called **Get CVE/NVD** and **Get PS**. These two buttons allow the selection of the data source as well as the associated model. **Get CVE/NVD** selects the provided demonstration dataset and loads the model for the NVD extraction. **Get PS** works the same way, but in contrast, it loads the resources for Packetstorm.

Clicking on the button may result in a short waiting time as the resources are being processed and the entity manager factory must be created. Once finished, the first entry is shown in the GUI and all fields display the extracted information.

10.2.2 Adding, Removing or Modifying Affected Products

The product table presents the affected products as well as their versions. However, in case of an incorrect/incomplete classification, the user may want to add, modify or remove an entry from this table. Therefore, clicking the **Add** button adds an empty editable row to this table. Moreover, by clicking **Remove** the selected row is removed from the table. To modify, the content of an affected product, a cell can be clicked and adapted. In order to apply the changes, it is necessary to hit the enter button.

10.2.3 Adding, Removing or Modifying References

This works the same way as modifying affected products.

10.2.4 Modifying Vulnerabilities, Precondition and Postcondition

The text fields for vulnerabilities, precondition and postcondition are editable. By clicking into the fields their content can be adapted. In case an additional entry shall be added, a semicolon (";") can be utilized as separator. The Weakness field is not editable, as the associated weakness is derived from the NVD and must only be contain existing CWE-IDs.

10.2.5 Assigning Values Relevant for Risk Management

This section explains the interaction with the **CVSS**, **Exploitability**, **Impact** and **STRIDE** components of the GUI.

It was decided that the **CVSS** field is read-only, in order to keep the correct value for decisions in the risk management process. The **Exploitability** and **Impact** values, however, are editable. This enables a reassignment of the values, in case the automatically derived value is too high or low in the eyes of the expert. Moreover, it enables the assignment of an **Exploitability** or **Impact** to entries that do not contain a CVSS.

As far as the field for **STRIDE** is concerned, not all NVD entries allow for a decision on a respective value. In the case of Packetstorm these values are not contained at all (at least not in structured form). Therefore, the assignment is mostly conducted manually.

10.2.6 (Re-)Generate Rule Text

When a new entry is loaded for display inside the GUI, the rule text is automatically generated. However, in case the content inside the affected product table is updated, the **"Generate Rule"** button must be clicked to update the **Rule text** field. The **"Rule text"** field itself is not editable as it shall only display the automatically generated rule text.

10.2.7 Storing the Vulnerability in the Database

Once all the content presented in the GUI has been reviewed and potentially adapted, a click on the **Save** button in the below the **Rule text** field triggers storing it in the database. Once the saving is complete, the next entry is loaded and displayed.