Dissertation

Towards Automated Advanced Vulnerability Analysis

by
Marian Gawron

Supervisors

Prof. Dr. Christoph Meinel

Chair Internet-Technologies and -Systems

Hasso Plattner Institute at University of Potsdam

March, 2019

Published online at the
Institutional Repository of the University of Potsdam:
https://doi.org/10.25932/publishup-42635
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-426352

Abstract

The identification of vulnerabilities in IT infrastructures is a crucial problem in enhancing the security, because many incidents resulted from already known vulnerabilities, which could have been resolved. Thus, the initial identification of vulnerabilities has to be used to directly resolve the related weaknesses and mitigate attack possibilities. The nature of vulnerability information requires a collection and normalization of the information prior to any utilization, because the information is widely distributed in different sources with their unique formats. Therefore, the comprehensive vulnerability model was defined and different sources have been integrated into one database. Furthermore, different analytic approaches have been designed and implemented into the HPI-VDB, which directly benefit from the comprehensive vulnerability model and especially from the logical preconditions and postconditions. Firstly, different approaches to detect vulnerabilities in both IT systems of average users and corporate networks of large companies are presented. Therefore, the approaches mainly focus on the identification of all installed applications, since it is a fundamental step in the detection. This detection is realized differently depending on the target use-case. Thus, the experience of the user, as well as the layout and possibilities of the target infrastructure are considered. Furthermore, a passive lightweight detection approach was invented that utilizes existing information on corporate networks to identify applications. In addition, two different approaches to represent the results using attack graphs are illustrated in the comparison between traditional attack graphs and a simplistic graph version, which was integrated into the database as well. The implementation of those use-cases for vulnerability information especially considers the usability. Beside the analytic approaches, the high data quality of the vulnerability information had to be achieved and guaranteed. The different problems of receiving incomplete or unreliable information for the vulnerabilities are addressed with different correction mechanisms. The corrections can be carried out with correlation or lookup mechanisms in reliable sources or identifier dictionaries. Furthermore, a machine learning based

verification procedure was presented that allows an automatic derivation of important characteristics from the textual description of the vulnerabilities.

Zusammenfassung

Die Erkennung von Schwachstellen ist ein schwerwiegendes Problem bei der Absicherung von modernen IT-Systemen. Mehrere Sicherheitsvorfälle hätten durch die vorherige Erkennung von Schwachstellen verhindert werden können, da in diesen Vorfällen bereits bekannte Schwachstellen ausgenutzt wurden. Der Stellenwert der Sicherheit von IT Systemen nimmt immer weiter zu, was auch mit der Aufmerksamkeit, die seit kurzem auf die Sicherheit gelegt wird, zu begründen ist. Somit nimmt auch der Stellenwert einer Schwachstellenanalyse der IT Systeme immer mehr zu, da hierdurch potenzielle Angriffe verhindert und Sicherheitslücken geschlossen werden können. Die Informationen über Sicherheitslücken liegen in verschiedenen Quellen in unterschiedlichen Formaten vor. Aus diesem Grund wird eine Normalisierungsmethode benötigt, um die verschiedenen Informationen in ein einheitliches Format zu bringen. Das damit erzeugte Datenmodell wird in der HPI-VDB gespeichert, in die auch darauf aufbauende Analyseansätze integriert wurden. Diese Analysemethoden profitieren direkt von den Eigenschaften des Datenmodells, das maschinenlesbare Vor- und Nachbedingungen enthält. Zunächst wurden verschiedene Methoden zur Erkennung von Schwachstellen in IT Systemen von durchschnittlichen Nutzern und auch in Systemen von großen Firmen entwickelt. Hierbei wird der Identifikation der installierten Programme die größte Aufmerksamkeit beigemessen, da es der grundlegende Bestandteil der Erkennung von Schwachstellen ist. Für die Ansätze wird weiterhin die Erfahrung des Nutzers und die Eigenschaften der Zielumgebung berücksichtigt. Zusätzlich wurden zwei weitere Ansätze zur Repräsentation der Ergebnisse integriert. Hierfür wurden traditionelle Angriffsgraphen mit einer vereinfachten Variante verglichen, die auch in die Datenbank integriert wurde. Des Weiteren spielt die Datenqualität eine wichtige Rolle, da die Effizienz der Analysemethoden von der Qualität der Informationen abhängt. Deshalb wurden Probleme wie Unvollständigkeit und Unzuverlässigkeit der Informationen mit verschiedenen Korrekturansätzen bewältigt. Diese Korrekturen werden mithilfe

von Korrelationen und Maschinellem Lernen bewerkstelligt, wobei die automatische Ausführbarkeit eine grundlegende Anforderung darstellt.

Contents

List of Figures

List of Tables

List of Algorithms

List of Listings

List of Equations

Chapter 1

Introduction

1.1 Current Situation

Nowadays, the security of computer systems receives continuously more attention. The systems are interconnected and integrated in nearly all fields, which results in an immense dependability on the functionality of the computer systems. For example, if the computer system of a doctor fails, the doctor cannot handle the patients anymore, because he cannot access their file and their details. Through this dependability arises the requirement that those systems should function properly at all time. Furthermore, it should not be possible to abuse those systems, because they contain confidential information or have access to crucial services. These requirements become even stricter if large companies are considered. Usually, those companies lose a large share of their profit and their reputation if a security incident occurs and their service is not available. In addition, one of their most valuable goods is the customer data, which is often the reason for their success. However, this customer data has to be stored securely and it has to be ensured that nobody is able to gain an unauthorized access. These facts contribute to the rising interest in security, because they also increase the attention of attackers since they could make a lot of profit with a successful attack. Thus, an estimation of the costs that were caused by cybercrime in the year 2017 results in an overall amount of 445 to 608 billion dollars [36]. Hence, it is not surprising that both, attackers as well as companies, focus on the security of their computer systems, since they can either make or save a lot

of money.

However, it is usually not trivial to increase a systems or a company's security. Modern computer systems have a high complexity, because they contain various applications, libraries, modules, or additional software. This large amount of different components often excels the manually manageable number of applications. Thus, it is difficult to create a comprehensive representation of the systems inner structure and even more difficult to retain the actuality and security of each individual component. The different applications and operating systems with their specific version allow a tremendous number of possible combinations, which might introduce additional interdependencies and increase the overall complexity of the system even further. These problems already affect a single computer and raise its complexity by a large amount, but if computer networks are considered, administration and maintenance become more and more difficult and intricate. Since computer networks include the interoperability and combination of various single computer systems and add the network configuration and network dependent functionality on top, the administration and security of those networks is a challenging goal for each administrator. Therefore, it is almost impossible to foresee all possible side effects and interoperability issues that can occur if hundreds of computers are interconnected in a complex corporate network with different subsystems. However, the security of those networks strongly depends on a preliminary evaluation of all possible combinations of inconsistencies that could finally lead to a weakness of the network. The complexity of this evaluation of the system's security requires an automated analysis.

Because of the past security incidents that have been published, many companies introduced logging capabilities in their corporate networks. Thus, it is possible to identify actions within the network with the goal to identify malicious activity. However, large companies, which invest the money to integrate a sophisticated logging infrastructure with a log management system or a security information and event management system (SIEM), usually have the problem of too many log events. The amount of log events can be controlled by an adjustment of the log level or the logging policy. This adjustment defines the events that should be logged and events that are omitted. However, an accurate

definition of the log level is difficult and the reduction of events is a trade-off with the comprehensiveness of events of a possible attack. Thus, the log level should be detailed enough to document the activity of a potential attacker, but this results in the documentation of the normal activity of the regular users as well. Therefore, the amount of logs reaches a volume that cannot be handled easily. In the last report from Gartner Inc., the requirements to current SIEM systems are estimated and divided into different deployment sizes. So, the number of events per second varies from 1,500 for small deployments to 25,000 for large deployments [21]. Nevertheless, even 1,500 events per second accumulate to a considerable number of events, which requires enhanced capabilities to process or manage the aggregated information. Especially, analysis on the logs is a challenging problem, because the overall amount of log data, which has to be processed, consumes too much performance. Thus, many analytic approaches can only work on a short time-frame or a limited set of pre-filtered events.

Another major impact factor is the user awareness. Several reports about security incidents begin with the loss of credentials of a valid account, because of some phishing email. Those emails deceive the receiver that his action is required for a specific service. Usually, some link to the service is included, which should be used by the receiver and directs him to a falsified page. If the user then types its credentials to login and perform the required action, the phishing website stores the gathered credentials and possibly redirects to the original service to remain stealthy. Thus, the credentials from the user can be misused for malicious activity. The number of phishing emails is ever increasing since 2015 and amounts to the same number as normal spam emails [33]. However, many trivial phishing emails aim at a broad target group and are easier to detect, since they contain language mistakes or other inconsistencies. Nonetheless, it is still profitable if only a small portion of users fall for these emails, since the effort to create those phishing emails is rather small. Another more targeted type of phishing uses additional information for the specific user, which can be gathered via social engineering techniques. Those spear-phishing emails are often used in advanced persistent threats (APTs). Famous examples of APTs that started with a spear-phishing attack are sykipot [66], ghostnet [14], or the RSA attack [34]. Thus,

many attacks often begin with an unaware user that clicks on a specially prepared link or executes a malicious program or script. Therefore, the awareness of users has to be enhanced through training, because a single unaware employee could introduce a malicious program or lose his credentials and the entire corporate network will be infected.

Besides the normal user awareness, it is necessary that the administrative staff of an IT infrastructure is aware of potential security weaknesses and disclosed vulnerabilities. As it was described before, many of the famous examples start with a phishing or spear-phishing email, but they often continue with an exploitation of detected vulnerabilities to establish a foothold in the IT infrastructure. This foothold should guarantee a long-term access, even if the affected user recognizes some strange behavior. Sometimes, the attackers use zero-day vulnerabilities, which are publicly unknown, as it was the case with the RSA attack [34]. These exploits are usually sophisticated and it is extremely difficult to detect or reconstruct these attacks. However, those vulnerabilities are also hard to find in the first place, which is the reason why those vulnerabilities are valuable and can be sold or bought for high prices. On the other hand, the majority of vulnerabilities that are exploited and lead to security incidents are already known but remain unfixed in the IT infrastructure. A report even estimated that 99% of the vulnerabilities that will be exploited by the end of the year 2020 are known by security professionals at the time of the incident [20]. This results in the fact that the majority of security incidents could be prevented by a preliminary vulnerability detection and analysis.

Furthermore, the overall security awareness of all users of the Internet would resolve several problems and minimize the attack surface of the corresponding systems. In addition to the system's security, it could also increase the difficulty to attack multiple systems and create a large number of bots to perform large scale denial of service attacks. Thus, the vulnerability analysis capabilities of public available services can contribute to the security awareness and the security overall. This was the reason for the publicly available vulnerability database from HPI [27], which provides vulnerability information and analytic capabilities. The HPI vulnerability database was already accepted by several users and received

friendly and positive feedback[1], such as "..I use the API of the HPI-VDB to perform request for current security updates of affected components via script.." Another user even states "I have discovered this offer and tried the functionality. It is very nice that there is another possible for systems diagnosis, even with an explanation of the vulnerabilities." One of the first feedback, which was received, described the HPI-VDB as "a super service, your attack detection, but how could the detected vulnerabilities be resolved?" This feedback resulted in the idea of a solution creation and integration into attack graphs, which will be explained later.

1.2 Significance of Vulnerability Analysis

As it was described before, the vulnerability analysis can be utilized to enhance the security of an IT infrastructure and prevent security incidents. However, the complexity of modern computer systems complicates this task and makes it challenging to perform a comprehensive detection. Additionally, the vast amount of information about vulnerabilities has to be handled and integrated into the detection process. This information is distributed in various sources over the Internet, as individual vendors, researchers, or governmental institutions release information about disclosed vulnerabilities. Vendor specific information about vulnerabilities can be found from Microsoft in the Microsoft security bulletins [40] for example. Certainly, some vulnerability databases are publicly available that already encompass many vulnerability information, such as the national vulnerability database (NVD) [46], Secunia [16], the open source vulnerability database (OSVDB) [50], or the open vulnerability assessment language (OVAL) [8]. But some of those vulnerability databases are already discontinued, for example OSVDB, or they include only a small set of vulnerabilities, such as OVAL or Secunia. Furthermore, security news portals, such as SecurityFocus [59] also publish information about vulnerabilities and in the case of SecurityFocus, it contains information about all published information of historic vulnerabilities. Thus, those portals could be used to find information for vulnerabilities as well.

[1]The mentioned quotes were extracted from emails to the HPI-VDB webadmin mail address

An additional source of information about recently published vulnerabilities is rich site summary (RSS) feeds, which are offered by some vulnerability research providers. The minimal and straightforward structure of the vulnerability information from [70] can be utilized to extract the required information.

Therefore, the vulnerability information are usually publicly available and can be extracted or gathered from several sources. Nevertheless, a remaining challenge is to process and interpret the information correctly, since several sources and publishers release the information in their proprietary format. Thus, a normalization of the included information is required to map the specific characteristics to a comprehensive data model. In addition to the normalization, it is necessary to convert some information from natural language to a machine-readable format, because the amount of vulnerability information and the number of different system characteristics require an automated approach to identify vulnerabilities. For example, the information in the vulnerability descriptions, which is usually included in all sources, is inapplicable for automated approaches, because it cannot be interpreted automatically. The vulnerability descriptions were originally not intended to be usable by automated approaches, but rather by human experts that are able to derive the necessary details from the natural language. Beside the descriptions, the available standards to identify the requirements and the impact of a vulnerability, as well as the standard to identify affected application should be utilized. Therefore, each vulnerability source requires a custom conversion for all characteristics. After the conversion into a machine-readable format, the information can be processed by automated analysis approaches, which is a fundamental requirement for the overall vulnerability analysis. An early idea for the normalization of vulnerability information was already implemented in [53]. This idea was developed further to also include the established standards and fully support the desired vulnerability analysis approaches.

Early ideas for an automated vulnerability analysis were already presented in [4], which proposes a set of characteristics for each vulnerability. These characteristics can be examined to determine the existence of the vulnerability. Another early approach was presented in [32], which introduces the idea of attack

graphs. To create the attack graphs the authors of [32] use a similar concept as the proposed conditions in this work. However, many previous ideas suffer from several limitations, such as the small number of considered vulnerability types in [4].

Currently, a popular possibility to detect vulnerabilities is the utilization of vulnerability scanners. One of the most famous tools to identify vulnerabilities automatically at the moment is Nessus [63], which is able to identify vulnerabilities on local machines or in larger networks. Therefore, Nessus runs several tests and examines predefined requirements to verify the existence of different vulnerabilities. The knowledge base that contains all these tests and definitions is built from more than 100,000 plugins, which will be used in the actual scanner. These 100,000 plugins cover around 45,000 CVE IDs and 28,000 Bugtraq IDs, which is a considerable amount of vulnerabilities [62]. However, the total amount of CVE IDs accumulates to more than 100,000 vulnerabilities, which means that Nessus does not cover a large fraction of vulnerabilities.

Thus, the necessity for automated vulnerability analysis is definitely required, because of the increasing complexity and the growing importance of modern IT infrastructure in most companies. A running computer network is a fundamental requirement to guarantee availability of services and the functionality of the company's communication. So, a malfunction could result in monetary losses as well as image losses for the publicity. Therefore, security mechanisms need to be in place to protect those valuable infrastructures and especially vulnerabilities should be resolved, since this is a precaution method to secure the network. The resolving of vulnerabilities implies the detection of those vulnerabilities in advance, which will be shortly discussed in the next Section.

1.2.1 Importance of the Application Detection

The ability to automatically detect vulnerabilities in IT infrastructure does not solely rely on the machine-readability of the vulnerability information, but rather on the identification of the infrastructures inner components. The inner components of the infrastructure include special devices, operating systems, or ap-

plications. Since vulnerabilities always affect a special version of a device, an operating system, or an application, it is a fundamental requirement to identify those components and examine whether one of the inner components is affected by a vulnerability. Certainly, the relation between the vulnerability and the application has to be interpreted and should be available in a machine-readable format. However, the relation does not provide any benefit if the affected software cannot be matched against all identified applications on the target system. Therefore, a comprehensive identification of all installed or running applications of the target system is the key requirement of the vulnerability detection. Thus, the analytic approaches that will be presented later are based upon the preliminary detection of all applications of a target IT system. This is also the reason why the application detection will play an important role in the vulnerability detection approaches and will be explained in more detail for each of the detection methods.

1.2.2 Passively Gather Information

The complexity of the application detection results in excessive and exhaustive scans of the target infrastructure. However, recent development in the field of security showed that SIEM systems get deployed in more and more corporate environments and the yearly investment into logging infrastructure continuously increases [21]. Therefore, an additional idea to collect information about vulnerabilities arises with the fact that insights about the components of an IT infrastructure are already available. Then, expensive scanning of the target infrastructure can be omitted and the influence on the performance of the system can be minimized. The idea to passively gather information about running applications in a network was already presented in [26], which introduces an approach to identify applications, especially web browsers, based on the network's traffic. However, the approach suffers from inaccuracy of the detection methods. The company behind Nessus also presented a passive scanning mechanism, which was integrated in a tool that tries to detect applications based on the network traffic [13]. Since the importance of the application detection does not only af-

fect the vulnerability detection but is also fundamental for other areas, such as inventory management, the field of service discovery was explored with different purposes. For example, running applications can be detected by analyzing the network traffic, as it was presented in [3]. Although inactive applications will not be covered by those detection methods, it can also be used to identify unnecessarily installed applications, which could be removed from important devices. However, in the case of vulnerability detection the fundamental challenge to identify running or installed applications could be solved by processing the available log data. This data could be either network traffic, as it was presented in several related approaches [26][13][3], or it can be gathered from the target machines directly. Furthermore, event logs of ongoing or past activity from the IT infrastructure could be used, as it is common practice that those log information are collected nonetheless. A remaining challenge would be the processing of those large amounts of data and the derivation of the application that produced the log event. In [31] a hierarchical knowledge base of regular expression was presented, which is able to process many events in real time and identify the original applications that produced the logs. Thus, it should be possible to utilize available information to reduce the impact of the vulnerability analysis on the target system. Additional insights and an implemented approach will be presented in Section 4.4.

1.2.3 Data Quality Assurance

Beside the different formats, the numerous sources of vulnerability information result in an additional challenge. The challenge is the processing of different information in the same vulnerability attributes. Therefore, decisions have been made that resolve those discrepancies or specify which information should be used. Thus, methods to verify vulnerability information have to be created and implemented into the system to allow an automated processing of the information with a high level of accuracy. Several research work have been investigated that also deals with automatic information derivation from textual information of vulnerabilities, such as [56]. Some vulnerability characteristics, which are

important for the analysis, can also be derived from different classes of vulnerabilities. For example, remotely accessible vulnerabilities all belong to a separate class if the CVSS attributes and especially the attack vector is considered. Thus, a classification of vulnerabilities can also help in assuring a high data quality, as it was also shown in [5]. Another work used data mining approaches to identify vulnerability characteristics from the textual information as well [69]. However, this research focuses on the classification of software bugs in either hidden impact bugs or regular bugs. The former class relates to vulnerabilities. Thus, the method which results in a bug classification of regular bugs and vulnerabilities can be applied to identify vulnerability disclosures in bug forums. The automatic derivation capabilities have also been applied to the field of identifying vulnerabilities, as it was shown in [29], which aims at the detection of vulnerabilities in the source code and in version control systems. Furthermore, the authors of [48] tried to predict vulnerabilities in software components based on a similarity score, which is computed in comparison to software components that are vulnerable. In another research work, the prediction of the time until the next vulnerability is disclosed for a specific program was investigated [71], which does not give insights in the details of the vulnerability. However, it could have been used for evaluating the risk of a utilization of that particular software. All of the prediction approaches suffer from poor results, as the authors in [71] have stated or the focus of the research work is more in the field of white-box and source code analysis, which is not considered for this work.

1.3 Thesis Contributions

This thesis strives to improve the current situation of security of modern IT infrastructures. It focuses especially on the detection and analysis of vulnerabilities in single computers or large networks. The challenges and requirements arise from a high complexity of the systems and a broad and widespread diversity of vulnerability information, which can be found in several sources. The individual contributions to the field of vulnerability research can be divided and assigned to the following topics.

Information Retrieval and Normalization The first challenge is to retrieve the widespread information from different providers. Therefore, several vulnerability databases, bug forums, or vendor information pages, such as NVD [46], Microsoft Bulletins[40], OSVDB [50], or Zero Day Initiative Advisories [70], have been integrated. The information from each source has to be extracted, but the different formats required additional effort. Thus, the normalization of the vulnerability information was necessary to convert the information into a common format. Then analytic approaches are able to process the information from different sources by using the common vulnerability model. The challenges and the solutions were also published in [22].

Vulnerability Classification with Machine Learning The different information sources often do not contain all desired information, but rather a subset of the vulnerability characteristics defined in the common vulnerability model. So, different approaches to automatically complete the information during the import of new vulnerabilities have been invented. The classification approach focuses on the CVSS attributes of vulnerabilities, which are particularly important for analytic approaches. The concrete approach will be explained in Section 2.3.3. The classification model derives the most important CVSS characteristics, namely the impact of confidentiality, integrity, and integrity and the access vector, from the vulnerability description. Security experts usually perform this process manually, which often results in a delay until the information is available. This delay could be mitigated by the automatic classification approach that was published and presented in [24].

Attack Graph Creation Furthermore, the vulnerability model was originally designed to contain machine-readable characteristics and allow an automated processing of the vulnerability information. Hence, preconditions and postconditions for each vulnerability have been created. Those conditions can be interpreted and evaluated automatically, but they are especially convenient for creating attack graphs. Since attack graphs usually visualize a possible path of an attacker, it arranges vulnerabilities in the network infrastructure according

to requirements and impact of their exploitation. The requirements for vulnerabilities are equivalent to the preconditions and the impact is equivalent to the postconditions. Thus, the chosen vulnerability model is well suited for attack graph creation. Two different approaches to create attack graphs will be explained and have been presented in [23].

Creation of Solutions and Integration in Attack Graphs Beside the creation of traditional attack graphs, another approach was designed to produce interactive attack graphs that contain information about the solutions for the detected vulnerabilities. This allows a direct assessment of possible countermeasures and their effectiveness. However, the information about possible solutions is not always available. Therefore, it has to be created and derived from the vulnerability descriptions. It turned out that the vulnerability descriptions often contain information about possible mitigation techniques for the vulnerability, but this information is only available in natural language. So, a transformation is needed that has to process the information automatically, because no human interaction is possible for this large scale of information. The approach was also presented and published in [23].

Automatic Detection Approach One of the key approaches in the field of vulnerability research is the detection of vulnerabilities in the first place. Because of the high complexity of the underlying IT systems and the variety and volume of vulnerability information an automated detection approach was required. This approach benefits from the vulnerability model and the direct association to the affected application or the affected component. However, the challenge to detect applications of the target infrastructure had to be solved and two different possibilities for different operating systems have been published in [22] and will be discussed.

Passive Vulnerability Detection Another detection approach was presented in [25]. This approach does not actively scan the target infrastructure but benefits from already available information from log management or SIEM systems.

It was shown that it is possible to reduce the impact of an exhaustive detection method especially for larger corporate environments. The approach has been implemented and the feasibility was shown in an experiment with real-world data from a large network in a research cooperation.

Publicly Available Database The before listed analytic approaches and data quality approaches have not only been invented but also implemented into a publicly available service [27]. The HPI vulnerability database (HPI-VDB) was originally created in 2012. It is continuously running and importing newly disclosed vulnerabilities. Furthermore, the service provides the possibility to utilize the described approaches. Therefore, the public version of the HPI-VDB allows registered users to perform an automated vulnerability detection or the creation of an attack graph for a given network description. An additional service that only considers the user's web browser was created to allow non-registered users to test the functionality of the HPI-VDB. The browser based vulnerability detection also has the advantage that it is not necessary to specify additional information about the target system. It merely uses the available information, which is sent along with each request.

1.4 Thesis Structure

The thesis is organized in six chapters. The first chapter introduces the topic and presents the current situation. The motivation for vulnerability analysis are discussed and the importance of the automatic approaches is emphasized. Then the contributions of the thesis are listed and the outline of the thesis concludes this chapter. The second chapter contains definitions and explanations of technical standards and technical terms that will be used in this work. The utilization of established standards allows an interoperability and a better integration with third party tools. Afterwards, the public available database is introduced [27]. Therefore, design decisions for the vulnerability model are explained and challenges in the implementation are highlighted. Several performance improvements will be discussed and the chapter is concluded by the explanation of the imple-

mented analytic features of the HPI-VDB. The following chapter illustrates use cases for the vulnerability information. The different scenarios benefit from the proposed vulnerability model and the specially generated conditions. It also highlights the practicability and applicability to real-world scenarios. The different use cases require a high data quality of the vulnerability information. Therefore, Chapter 5 introduces different methods to ensure the data quality. Hereunto, the formerly introduced standards will be considered and the completion or validation of the attributes in the comprehensive vulnerability model is explained. Finally, an adjustment to the currently used vulnerability scoring system is presented, which could increase the usefulness of the current system. The last chapter concludes the work and summarizes the findings and investigations, which have been achieved in the area of vulnerability analysis.

Chapter 2

Glossary and Technical Standards

A comprehensive security analysis always starts with a definition of the analysis domain and all possible scenarios that should be investigated throughout the analysis itself. Thus, it is necessary to also define the fundamental basics of security to create common knowledge base. Therefore, in this chapter I want to introduce some basic terms, abbreviations, and common standards that will be used in the following work.

2.1 Security Goals

When IT security is considered one usually talks about security goals that have to be achieved for a process, a system, or a protocol. This division is required, because security itself influences every aspect of IT systems and therefore has a vast variety and many different shapes. The subdivision of IT security results in many individual security goals that have to be considered and weighted according to their importance for the underlying system. For example, the confidentiality of data is important for cryptography, whereas anonymity of the communication participants is usually inappropriate.

The variety of security goals resulted in the classification of those goals into basic and additional security goals [15]. The different classifications assign either three or four goals to the basic security goals. The *non-repudiation* is sometimes considered as an additional goal and sometimes as a basic goal. However, the other three common goals are always considered as basic goals, which are namely

confidentiality, *integrity*, and *availability*. They are often referred to as the CIA goals or the CIA triad. Additionally, experts often use these three goals as a boundary to place systems and protocols in the so-called security triangle [38].

2.1.1 Confidentiality

In the official standard of RFC-4949 [61] *data confidentiality* is defined as "the property that data is not disclosed to system entities unless they have been authorized to know the data." This definition could be generalized and applied to information in general, as the concept of confidentiality is much older than the Information Technology. Confidentiality describes the property that information is only accessible to authorized systems or individuals. In the context of confidentiality accessible refers to the ability to read, interpret, or understand the information.

2.1.2 Integrity

The before mentioned standard of RFC-4949 [61] contains a definition for *data integrity* as well. It defines *data integrity* as "the property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner." If we only consider *data integrity* on computer systems themselves, this definition is sufficient and could be realized with access control lists or similar mechanisms. If one considers `data integrity` in the context of network connections, the possibilities and applicable mechanisms are limited. It is not possible to protect data against unauthorized change or loss during the transmission as the connection medium cannot be controlled. Thus, it is only possible to detect and sometimes correct these unauthorized or accidental modifications. Nevertheless, the *data integrity* has a high impact on a systems security, since the deliberate modification of data can change the control flow of a program and result in the corruption of the entire system.

2.1.3 Availability

The third basic security goal is *availability*, which is often considered as the most important security goal by service providers. Service providers usually highly rely on the accessibility of their service, as customers tend to move away if the requested resource is not available. The RFC-4949 [61] defines *availability* as "the property of a system or a system resource being accessible, or usable or operational upon demand, by an authorized system entity, according to performance specifications for the system [...]". The *availability* should ensure the timely, reliable access to data and services for authorized entities.

2.2 Additional Terms

Beside the above defined security goals, the research in vulnerabilities relies on several other established terms. I want to mention and explain the most important terms in the following to create the common foundation for the following work.

2.2.1 Software Bug

A software bug is an erroneous part of the application that usually originates from a human mistake or inaccuracy in the implementation. The bug itself does not have to have critical effects on the security, but depending on the location and type of the bug, it could provoke malicious behavior of the application.

2.2.2 Vulnerability

The official standard in the RFC-2828 [60] defines a vulnerability as "a flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy". Furthermore, the IETF states that most systems might have vulnerabilities, which does not directly mean that those systems should not be used. Not every vulnerability leads to an attack and some vulnerabilities might be hard to successfully attack.

Nevertheless, if the attacks are well understood and the vulnerable application is widely distributed, the possibility of a successful attack rises together with the benefit for the attacker.

2.2.3 Exploit

Generally, an exploit describes the act of abusing a vulnerability and provoke an unintended behavior of the affected program. The result of a successful exploitation could be a malicious action, which is specified in the payload of the exploit. The exploit itself can be a piece of software or a set of instructions to misuse the flaw. Furthermore, a multitude of exploits can be found in the exploit-db [49], which currently contains more than 38.000 exploits.

2.3 Common Standards

In the following, I want to introduce some standards that are well established in the domain of vulnerabilities and security analysis. I used these standards in the presented work to guarantee compatibility with existing solutions and services. Furthermore, the established standards should allow the comprehensibility, reusability, and interoperability of the approaches.

2.3.1 CVE

One additional important characteristic of a vulnerability is the *identifier*, which is used to refer to the vulnerability in different sources. As it was already described, information about vulnerabilities are distributed over the Internet and can be found in different locations, which include forums, patch notes, vendor pages, or public vulnerability databases. Most of the publicly available sources use proprietary identifiers in their databases. However, they often include an additional identifier to refer to the vulnerability across different platforms. This well-established standard is the *Common Vulnerabilities and Exposures*(*CVE*) [42]. The CVE identifier is a well-structured format and contains

the year as well as a running number to refer to a specific vulnerability. The running number was recently extended to contain more than four digits to allow a larger number of reports about vulnerabilities. The management and assignment of CVE identifiers is organized by the National Institute of Standards and Technology(*NIST*) of the United States. The NIST publishes information about the reported vulnerabilities in their vulnerability database *NVD*(National Vulnerability Database)[46] and assigns CVE-IDs to the vulnerabilities. Additionally, large software vendors can also acquire ranges of CVE-IDs for vulnerabilities that might be detected in their software products.

2.3.2 CPE

As it was described a vulnerability describes a weakness in a system's design or implementation. Usually vulnerabilities arise in applications with a specific version and get fixed in a later version of that application. Thus, vulnerability analysis heavily relies on the ability to identify the application itself and the affected version. This identification has to be performed with a high level of detail as a small bug fix that resolves the vulnerability could be indicated with a minor version change. The *Common Platform Enumeration*(*CPE*)[6] is tailor-made for this task. The NIST also manages a dictionary with a large set of CPE-IDs, which are used in the vulnerability definitions of the NVD. The structure of a CPE-ID follows a predefined schema that also allows the construction of CPE-IDs if the required application is not already listed in the dictionary. The CPE specification was updated to version 2.3 in August 2011 [11]. The previous version was widely used and integrated into my work to allow backward compatibility.

Listing 2.1: CPE Specifications

```
CPEv2.2:   cpe:/o:apple:mac_os_x:10.9.5
CPEv2.3:   cpe:2.3:o:apple:mac_os_x:10.9.5:*:*:*:*:*:*:*
```

Both specifications start with "cpe" and use the ":" as a delimiter. The most important fields are the type, the vendor, the application name, and the version number. The type could be either "a" for application, "o" for operating system, or "h" for hardware. Additional information, such as language pack,

platform specification, or service pack can be specified in corresponding fields at the end. As it is illustrated in the example in Listing 2.1 the new version requires the specification of wildcard characters for each attribute, whereas unused attributes could be omitted in the old version. This omission results in problematic artifacts if several parts in the middle have been completely left out, the remaining attributes were placed at wrong locations in the CPE-ID. For example, it could be possible that the language identifier is placed at the position of the architecture part. Therefore, the accurate parsing of the old version of the CPE identifiers was difficult and required more effort.

2.3.3 CVSS

Another established standard is the *Common Vulnerability Scoring System* (*CVSS*), which is used for weighting and to assign severity values to the different vulnerabilities. The CVSS was presented in 2006 [37] and was the first schema to rate vulnerabilities and their possible impact to the system. The current version of the Common Vulnerability Scoring System is 3.0 and was released in December 2014.

Score

Vulnerabilities are classified in different groups and receive numeric scores in the range of 0 to 10, with 10 as the highest severity. This numeric number is called *CVSS Score* and it is also assigned by the NIST that manages the NVD [18]. The scoring mechanism itself is publicly available. So everybody could reconstruct the CVSS-Scores and also assign CVSS-Score on its own. Since this rating schema also evolved, it is common to specify temporal and environmental metrics in addition to the base score. This differentiation allows a monitoring of exploitation and status upgrades for each vulnerability. However, the only mandatory properties are the base attributes and the base score. The calculation is strictly related to the values of the CVSS Vector, which will be explained in the following Section.

Vector

The mandatory part of each *CVSS Vector* is the base vector, which defines characteristics for the minimum set of attributes of a vulnerability. A successful evaluation and a severity ranking of a vulnerability is only possible if at least those attributes are known. The base vector of version 2 contains six attributes, which are the *access vector*, the *access complexity*, the *authentication* and the basic security goals *confidentiality*, *availability*, and *integrity* in this specific order. Each of these attributes can have three different values. The most important attributes are *access vector*, which indicates the connectivity an attacker has to have, and the three security goals. The range could be one of "local"(`L`), "adjacent network"(`A`), or "remote network"(`N`). The attributes for each of the basic security goals contains information about the impact on this specific security goal. So, each goal could have the values "not affected"(`N`), "partially affected"(`P`), or "completely affected"(`C`). This information is especially helpful in the estimation of consequences of a successful exploitation of a vulnerability. A sample vector was illustrated in Listing 2.2.

Listing 2.2: CVSS Vector

```
CVSSv2 base vector:
                                        AV:L/AC:M/Au:N/C:N/I:P/A:C
CVSSv3 base vector:
                                CVSS:3.0/AV:N/AC:L/PR:H/UI:N/S:U/C:L/I:L/A:N
```

The CVSSv3 vector contains the three security goals, information about the attack vector, and information about the attack complexity as well. The authentication indicator was replaced with user interaction and privileges required. Furthermore, another indicator for the scope was appended. Most of the values of the attributes remain, but one important addition was the value *physical* to the attack vector. A sample base vector is illustrated in Listing 2.2.

2.3.4 CWE

The *Common Weakness Enumeration*(CWE) [45] provides the possibility to classify vulnerabilities. The information is managed by the Mitre corporation [43] and it is organized in a hierarchical form. This structure helps to arrange vulnerabilities according to the underlying type of weakness. For example the "classical buffer overflow" has the *CWE-ID 120* and is the child of *CWE-ID 119* ("Improper Restriction of Operations within the Bounds of a Memory Buffer"). This relation represents the connection between a buffer overflow and the missing or wrong restrictions for bounds in a memory buffer, as a buffer overflow is only possible when the check for bounds of the buffer is incorrect or missing. The *CWE-ID 119* in turn is the child of *CWE-ID 118* ("Incorrect Access of Indexable Resource"), which is the superset of improper restrictions to boundaries of a memory buffer. So, one could also deduct further requirements and properties of a vulnerability from the relation to its CWE-ID, which could be interpreted as the type or family of vulnerabilities. In addition, the CWE-ID can complete the overall picture of a vulnerability, since not all individual properties are represented in the description of a vulnerability. Another interesting point is to identify relations between different CWE-IDs apart from parent-child relation that might allow the prediction of chaining vulnerabilities or their types that appear frequently.

2.3.5 OVAL

OVAL is the *open vulnerability assessment language* and it is maintained on the *OVAL* website [8]. The content is provided and managed by the already mentioned Mitre corporation [43]. It offers possibilities to assess and report on the machine state of a computer system. This is especially helpful if the inner state in terms of installed applications or vulnerabilities is considered. Thus, *OVAL* is able to create an overall list for the inventory of a computer system, which can be used by other approaches that rely on this detailed information about installed applications. Additionally, *OVAL* also proposes definitions and tests for vulnerabilities. Therefore, it also relies on the active community to extend

and provide the available tests and definitions to find installed applications or available vulnerabilities. However, the amount of vulnerabilities reveals the lack of several known vulnerabilities.

2.3.6 CAPEC

CAPEC is the *common attack pattern enumeration and classification* [65] and a standard to identify and address attack patterns. The commonly known attacks receive an ID to be referred to and to be identified for better analysis. The identifiers are managed and maintained by the Mitre Corporation [43]. The ability to identify possible attacks for a vulnerability might also result in a more accurate definition of the postconditions of a vulnerability. For example, *cross-site scripting* (XSS) has the CAPEC-ID 63 and is the parent of *reflected* and *stored* XSS and the child of *code injection*. When a vulnerability is connected to the CAPEC-ID 63, it is possible to deduce countermeasures immediately, since it is related to *code injection*, which could often be mitigated by input sanitization.

Chapter 3

HPI Vulnerability Database

3.1 Introduction

The HPI vulnerability database [27] was designed to collect vulnerability information from various sources and unify the information. Since the number of vulnerability reports and vulnerability discoveries increased over the years, an automatic processing technique is necessary. Thus, the information is converted into a machine-readable format that allows an automatic processing. The original purpose was to serve as a knowledge base for other projects and to create the foundation for vulnerability analysis. The first version of the database was developed in 2012 and has evolved since. The basic idea to serve as a knowledge base remains but several analytic approaches have been built around and implemented into the database as well.

3.2 Model and Format

The original design and the first data model for vulnerabilities was created in the initial development of the HPI vulnerability database. At this time, the vulnerability itself only contained a subset of attributes that are now available. The current design is illustrated in Figure 3.1. The CVSS attributes were extended and additional timestamps as well as identifiers to other standards have been added. During the design phase the identifier was created to contain any unique

string, which allows the combination of different IDs, such as Bugtraq ID [59], CVE-ID, Microsoft Bulletin [40].

Humans can use the description to explore the details of the vulnerability. Usually the description is always present in various sources, as most of the reporting procedures, such as the report form of Carnegie Mellon University [7], require a detailed description of the discovered vulnerability for reconstruction and investigation. This description is kept to allow a manual investigation of vulnerabilities in the HPI vulnerability database as well.

Furthermore, several references and third party IDs are used to link to external information and allow users to follow and investigate these third party resources. The CWE-ID can be especially used to include information about the vulnerability type or deduct additional characteristics of the vulnerability, as it was explained in Section 2.3.4.

Finally, each vulnerability contains information about its severity. Since the most commonly used schema CVSS, which was described in Section 2.3.3 allows an assignment of severity scores in the range of 0 to 10, that information was included as well. The CVSS information do not only include the score, but the information about the impact, violation of security goals, and the abbreviated form of all additional attributes in the CVSS Vector.

Besides these basic properties of the vulnerabilities that are directly included in the vulnerability data model, the database also contains related characteristics. As it is also illustrated in Figure 3.1 those features namely are source, CPE-ID, preconditions, and postconditions that could be related to more than one vulnerability.

The source is the most trivial characteristic, since it is used to save details about the origin of the vulnerability information. Additional sources can be specified and will be crawled periodically. This specification has to be performed manually and for some sources, it might be necessary to implement the reader interface or adjust an existing reader. Then, it is also possible to specify a reliability score that will be used when information from more sources have to be merged.

An important related attribute is the CPE-ID. As it was explained in Sec-

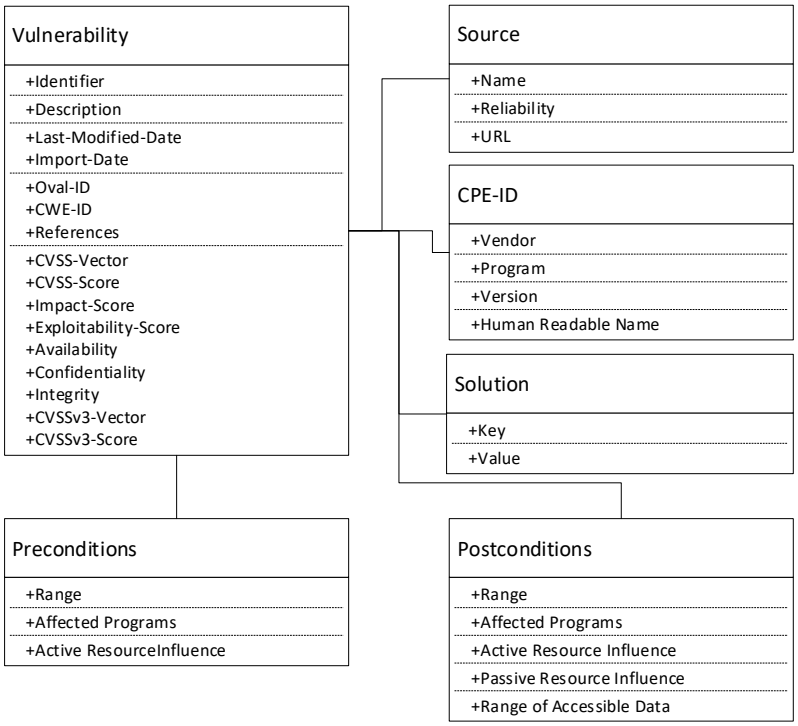


Figure 3.1: Vulnerability Model

tion 2.3.2, the CPE-ID can be used to identify software components with a high level of detail. Thus, the CPE-ID is used to identify the affected application and provide additional information about it, such as the vendor, the application name, or version information. Since the existence of the affected application is the main requirement for a vulnerability, this information will also be used as a foundation for the preconditions later. The fact that the existence of the application is the main requirement, resulted in the direct relation of the application to the vulnerability, although the connection could also be found in the preconditions. Nevertheless, several use cases and different analytic approaches showed that a filter or search for installed applications is usually one of the first steps of users or security specialists. So, a direct relation between affected application and vulnerability reduced the workload during analysis and data exploration. However, the relation in the preconditions and postconditions is still handled via

an internal ID of the CPE-ID object. This allows a fast processing of queries for CPE-IDs and maintains the relationship in the conditions while guaranteeing the consistency in the data. While monitoring the usage of the HPI-VDB it was observed that most users query the database for vulnerabilities to a specific application. This later led to the program stack functionality, that will be explained in Section 3.5.2.

The last related attribute, which is explained before introducing the two conditions, is the solution. The solution could have different characteristics based on the origin of the information. It could be a special reference, if the solution is directly extracted from third party information about the vulnerability. Then, the solution contains a link to this information. However, the solution could also be generated during the analysis of the vulnerability description. This analysis will be described in Section 5.2.2, as it is one of the major improvements of the HPI-VDB to automatically extract information from the textual description.

The two conditions that were mentioned before are used to guarantee machine-readability of the vulnerability information. The conditions are structured as sets with logical operators. The different members of each set can be combined using the logical operator that is specified in the set. The resulting value is a binary representation of the degree of fulfillment of the conditions. So, if the logical combinations of all members of a condition results in a positive value, the condition itself is fulfilled. The operators for the conditions are usually one of "or" or "and". When a condition contains the "or" operator a single member of the condition has to be fulfilled to fulfill the overall condition, whereas the "and" operator indicates that all members have to be fulfilled. Additionally, the conditions are organized in a hierarchical format, which means that a member of a condition can be a set on its own. The default parts of the precondition are usually the ability to produce input to the application, the required access range, and the existence of the application itself. In most cases the vulnerabilities can be found in several applications, which would lead to a set of individual applications that contains the "or" operator. The "or" operator is used when it is irrelevant which of the mentioned applications is running, as long as one of them is existent. On the other hand, it is also possible to create conditions that require

a combination of applications to be in place. For example if a vulnerability could only be misused in an application when it is executed on a Windows machine, a condition could be built with a set of the application and the Windows operating system connected with an “and” operator. The postconditions extend the attributes of the preconditions by the impact of the vulnerability on the data of the affected machine. Therefore, the postconditions consist of the applications, the range, the program influence, the data, and the data influence. There are some special cases that result in a different characteristic of the before mentioned attributes applications, program influence, and range. If the vulnerability affects the integrity of data, the attacker is able to read, write and delete data on the target system. The attacker is usually able to change data in a way that allows a capturing of the entire system. This means that the host itself is infected and it is not trustworthy anymore, since it could be used for additional attacks. Furthermore, the range of the attacker might change or another range could be added if the attacker is able to use this machine as a Pivot point to reach deeper into the network. The program influence can change when denial of service(DoS) attacks are considered since these attacks affect the existence of the application.

Listing 3.1: Example preconditions of Vulnerability CVE-2014-0160

```
<set operator="and">
 <set operator="or">
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1:beta1"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1:beta2"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1:beta3"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1a"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1b"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1c"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1d"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1e"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1f"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.2:beta1"/>
 </set>
 <prop key="program_influence" value="input"/>
 <prop key="range" value="remote"/>
</set>
```

For example Listing 3.1 illustrates the preconditions for the Vulnerability CVE-2014-0160, which is better known as the “Heartbleed” bug. This precondition

specifies OpenSSL of a specific version, a remote range for an attacker, and the ability to send data to the application. This vulnerability could be used to reveal confidential information, as it was prominently discussed in the public. This results in a violation of the confidentiality and in the postconditions, which can be found in Listing 3.2.

Listing 3.2: Example postconditions of Vulnerability CVE-2014-0160

```
<set operator="and">
 <set operator="or">
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1:beta1"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1:beta2"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1:beta3"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1a"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1b"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1c"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1d"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1e"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.1f"/>
  <prop key="application" value="cpe:/a:openssl:openssl:1.0.2:beta1"/>
 </set>
 <prop key="program_influence" value="input"/>
 <prop key="data" value="any"/>
 <prop key="data_influence" value="read"/>
 <prop key="range" value="remote"/>
</set>
```

The postconditions show that the application is still in place, the application also accepts input, and the range of the attacker was not modified either. Nevertheless, the additional impact on the data of the system is represented by a *data_influence* attribute which refers to the readability of data. Therefore, the postconditions represent the confidentiality loss and the possibility to read information from the target system.

3.3 Information Retrieval and Normalization

At the time when the HPI-Vulnerability Database was created, the vulnerability information, which have been publicly available are widely spread in the Internet. So, information could be found from the individual vendors, such as Microsoft with their Security Bulletins [40]. Additional information was provided by ex-

isting and publicly available vulnerability databases, such as SecurityFocus [59], National Vulnerability Database [46], or Open Source Vulnerability Database (OSVDB)[50]. The OSVDB is a special candidate, since it was discontinued in April 2016 and the service was shut down permanently. Nevertheless, in the beginning information from OSVDB was used as well, to enrich existing entries and extend the number of vulnerabilities. Other resources that are incorporated are the Exploit database [49] with almost 40 thousand exploits archived. For the HPI-VDB, vulnerability identifiers can be extracted from the exploit and it could be used as a third party resource. Some vulnerability databases offer exports and dumps for their data. For example, the National Vulnerability Database provides dumps for every year and another file for recently published vulnerabilities that is updated on a daily base. In addition, other providers offer RSS feeds that contain information about recently added vulnerabilities, such as the RSS feed from Zero Day Initiative[1]. These feeds often contain incomplete information and the format of the feed varies greatly. However, an advantage is that the information is already available even if the vulnerability was not completely analyzed yet. For retrieving vulnerabilities from the RSS feeds and from sources that do not provide exports or dumps of their data, crawlers are used to gather the information. For the sources that offer dumps, the data has to be retrieved in the same schedule as the providers update their data. In addition, a check for the Last-Modified header of the retrieved file is done and the result is compared against the stored timestamp from the last update of this source. This latest Last-Modified timestamp is always persisted in the database to be available for future checks. The stored timestamp can also be used to ensure that the information source is only queried after a certain threshold, to prevent continuous and extensive querying. Thus, a reliable mechanism to verify the necessity of an update is used to incorporate latest changes to the vulnerability information.

Then the information is normalized to match the defined structure of the HPI-VDB, which can be also seen in Figure 3.1. Since not all information is available in all sources, the model only requires a few attributes by default. To create a new

[1] http://feeds.feedburner.com/ZDI-Upcoming-Advisories

vulnerability, one has to specify information about the vulnerability identifier and the description, which are available for most sources. The additional analytic information, such as CVSS attributes, can be added later and will be needed to generate the conditions. If the CPE-ID is not available, it is possible to extract information from the description which might point to a CPE-ID in the CPE dictionary.

Another approach to find information about vulnerabilities was presented in my research [24], which introduced a new method to extract CVSS information of vulnerabilities from their descriptions automatically. This approach is based on machine learning and will be explained in more detail in Section 5.3. However, the idea to use this method to identify whether a text contains information about vulnerabilities can also result in an automatic approach to find information in the first place. In this work, different machine learning approaches have been presented to extract CVSS attributes from textual descriptions. Therefore, these methods work on arbitrary textual input and extract information, such as attack range or the violation of basic security goals. If these attributes can be extracted from a text, it is likely that the text contains information about vulnerabilities in general. Therefore, it can be used as an additional source for vulnerability information.

When the source for additional vulnerability information was specified, the already mentioned reader interface has to be implemented or adjusted. This adjustment is necessary to allow the processing of the information from the custom structure of the new source. Therefore, the reader interface defines functions and normalization capabilities that should be fulfilled by the reader of a new information source. In the case of XML exports or RSS feeds the reader could benefit from already existing libraries, such as etree from lxml[2], or feedparser[3]. Then the reader has only to map the specific fields from the parsed XML/ RSS model to the vulnerability object and populate the information for each entry. When the information is only available on websites, the reader has to be customized and the content of the site has to be parsed to extract necessary information. In

[2]https://pypi.python.org/pypi/lxml/4.1.1
[3]https://pypi.python.org/pypi/feedparser

that case libraries, such as scrapy[4], have been used to simplify the parsing and navigate to desired information based on the document object model (DOM) tree of the page.

Then, the extracted information is normalized and official standards are used whenever it is possible. So, the identifier is usually populated with the corresponding CVE-ID, if the information is available. Otherwise, the identifier of the source can be used for the time being. Later if the information is correlated with additional information sources, the identifier could be replaced. The description is not modified as it is not necessary for the analysis procedure and there is no standard for human readable descriptions. Additionally, third party sources and references are extracted and then transformed to a key-value pair, where the key represents the name of the source, which is usually the domain of the host of the information. The value contains the full link to the information and it is used to maintain the link to the reference in the vulnerability later. The treatment of the CWE-ID 2.3.4 is similar to the treatment of the identifier. However, if the information is not present the attribute will not be populated until information is found in another source or in a later version of the vulnerability information.

Another special attribute that is normalized is the information about affected software of the vulnerability. The CPE-ID was already explained in Section 2.3.2 and this schema is also used by many sources and public vulnerability databases, such as NVD [46]. Nonetheless, the HPI-VDB also integrates sources that do not contain the CPE-ID or that did not feature the CPE-ID in the past. Thus, the solution was to extract the information about the application from the description. Since the information about a vulnerability does not provide any benefit if the affected program is not mentioned, the description usually contains the application name. This allowed humans to understand the information about the vulnerability and correlate the information with the mentioned application. However, for the automatic integration an extraction procedure has to be performed to find the application name in the description. Therefore, the CPE dictionary was used to perform look-ups for specific applications that are already registered by the National Institute of Standards and Technology (NIST),

[4]https://scrapy.org/

which maintains the CPE dictionary. The extraction of the application information is achieved via a regular expression. Since the descriptions for vulnerabilities has to be understandable by most humans, it also follows some basic rules in the wording. It turned out that the application information is usually preceded by the preposition "*in*" or the conjunction "*and*".

Listing 3.3: Sample regular expression to identify application information

```
(?:^| in |, and | and |, )(.*?) (?:before|prior to)
(?:^| in |, and | and |, )(.*?) allows
```

After the software was specified there is a word indicating that this "*allows*" a specific action or that this software "*before*" or "*prior to*" a specific version is affected. In the latter case the "*before*" also indicates that a longer list of applications with a lower version number are affected. This list is not directly specified in the description but the regular expression in the first row of Listing 3.3 could be used to identify these cases. These observations have been used to generate regular expressions that identify the program part in the description. Two sample expressions are can be seen in Listing 3.3. This approach is similar to the proposed approach to automatically create solutions for vulnerabilities that is explained in section 5.2.2.

Another approach was to use the semantic background of the description. Therefore, the text is transmitted to the DBpedia spotlight api [12] and annotated there. It turned out that the requirement to process several hundred to a few thousand descriptions in a reasonable time could not have been satisfied. The investigation showed that the bottleneck is the processing of the description, which has to be done to create and correlate CPE objects for the vulnerability. Nevertheless, if the import process has to wait nearly 1 second for a description to be parsed, including the request and response from the API, the required time would increase drastically. In addition, the accuracy of the identification is not as high as expected. Moreover, the approach was unable to deal directly with large lists of applications, which are not described in the description. Especially vulnerabilities with a high number of affected applications, such as the recently discovered "Spectre" vulnerability (CVE-2017-5715 and CVE-2017-5753) with

1065 affected CPE-IDs each or some Chrome vulnerabilities that affect all previous versions (e.g. CVE-2017-5117) with 3756 affected CPE-IDs. These special cases do not contain all affected components in their descriptions but state that components before a specific version are affected. Certainly, the approach could be combined with a small script that checks for those specific terms, such as "*before*", or "*prior to*", but then it is again necessary to specify all these special cases manually, which reduces the independence and universality of the DBpedia approach. Thus, the approach could not be applied and relied on solely, but it could have been used as an additional classification procedure and a combination of the results from regular expressions and DBpedia classifications was planned. In the end, the low accuracy and the additional overhead in processing time were strong arguments to dismiss the DBpedia approach.

3.4 Performance Improvement

During the development of the HPI-VDB several challenges arose, which are tied to the amount of data and strongly related to the usability of the database. Since the database was designed to be accessible by the public, the usability and responsiveness was always a strong requirement. The analysis of user actions and requests to the HPI-VDB showed additional room to adjust the model and focus on specific jobs, such as searching, or import.

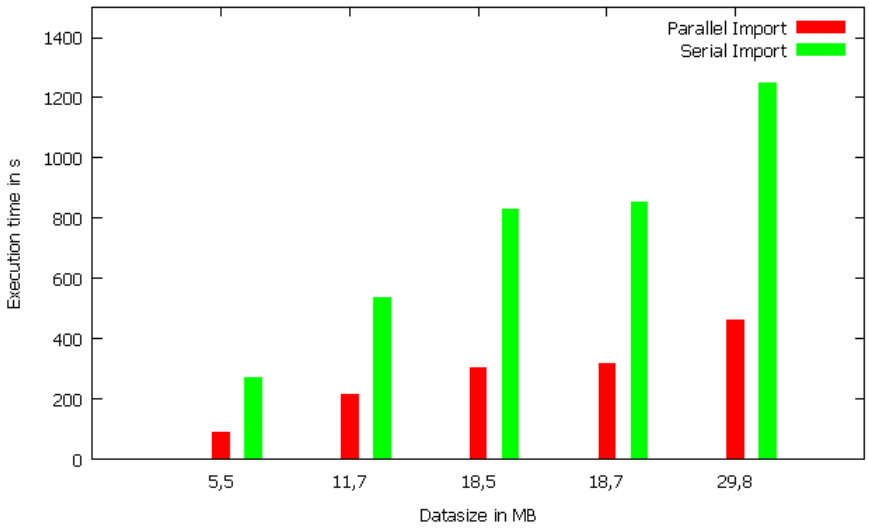
The first adjustment was an architectural modification to increase the performance for the majority of requests, which were observed from the logs. The decision to add an additional CPE object to each vulnerability resulted from the massive amount of searches based on program name. Originally the CPE-ID, which was explained in Section 2.3.2, was only specified in the preconditions of a vulnerability. Since the major requirement for a vulnerability is the existence of the affected application, the CPE-ID is the integral part of the preconditions. However, this resulted in the necessity to query the preconditions and reconstruct the sets to find all connected CPE-IDs for a vulnerability. Since the CPE-ID or multiple CPE-IDs are part of a set in the preconditions, it was necessary to obtain the preconditions first. Then, it is checked whether the preconditions directly

contain a CPE-ID or whether a set of applications is listed. In the latter case, this set has to be retrieved and each entry can then be returned, as it would be the case with the example preconditions in Listing 3.1. Although, this top-down description to find CPE-IDs for a given vulnerability illustrates the additional steps that have to be performed, the approach to find vulnerabilities for a given CPE-ID works in a similar manner. The only difference is that the relationship which is used to retrieve the property set for a specific member is *member_of*. Therefore it requires at least 2 additional retrieval steps to find a vulnerability for a given CPE-ID. Thus, a new CPE-ID model was introduced and connected directly to the vulnerability model via a many-to-many relationship. This means each vulnerability can have multiple CPE-IDs and each CPE-ID can have multiple vulnerabilities. This adjustment allows a direct query for a CPE-ID and receive the connected vulnerabilities without additional interaction with the preconditions. The preconditions were modified to not contain the value of the CPE-ID anymore but a reference to the corresponding CPE-ID object to avoid a duplication of information and ensure the consistency of the data.

The other improvement for performance reasons is related to the import. Since the database is updated on a daily base and different sources are incorporated, the import capabilities should allow the information integration in a reasonable time. The solution to this problem could have been a parallel import. The idea to use multiple threads or processes to import new information was derived from the observation that the majority of time to import a new vulnerability is used for the creation of preconditions and postconditions. The other attributes can often be extracted directly from the information source. So, based on the violation of security goals and the other mentioned CVSS characteristics special conditions are generated. These conditions contain the range, which can usually be directly extracted, but the lookup and combination of the CPE-IDs consumes a considerable amount of time. The idea was to run the condition generation in parallel and synchronize the access to the database, meaning the different write and lookup operations. The problem was that the CPE-IDs in the conditions was initially a key-value tuple and therefore not bound to any object. Therefore, the conditions could be generated independently and persisted in bulk requests.

With the parallel approach, it nearly speeds up the process by the number of cores that are available. The performance improvement from the parallelization is visualized in Figure 3.2.

Figure 3.2: Parallel Import Performance with 4 cores



However, the model adjustment to handle CPE-IDs as objects and use the object identifiers in the conditions requires additional lookup operations in the database. The read operations for the existence of a specific CPE-ID could be parallelized, so the parallel approach could deal with this additional lookup operations and achieve similar results as shown in Figure 3.2. However, additional tests revealed that the import is not capable to handle all cases successfully. For example cases, when a new CPE-ID object need to be created by multiple threads at the same time, cannot be processed. The initial lookup for the CPE-ID would yield that the CPE-ID is not listed in the database. Therefore, there is the necessity to create the CPE-ID object. However, when the threads obtain the lock to persist the results in the database the first thread is able to create the object, but already the second thread is not allowed to create the CPE-ID object, since each CPE-ID should only be listed once. Thus, the parallel import capability was discarded in the productive environment.

Nevertheless, a similar parallelization approach was implemented for the CPE-

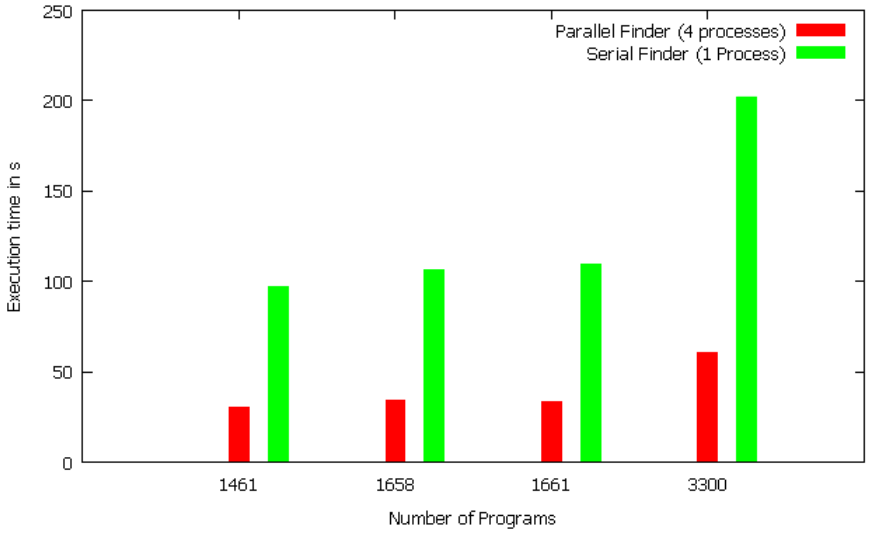
ID lookup in the CPE dictionary, which will be used for analytic approaches that are explained in Section 3.5.1. The problem was to find the correct CPE-ID for a subset of the required attributes. Usually, not all relevant attributes of a program could be detected or are returned by software detection tools. Thus, the analytic approaches have to handle cases when only a subset of attributes, such as name and version of a program, are available. Then, the lookup operation has to search through the CPE dictionary and find suitable candidates that have a match in the given attributes. Therefore, in the end the entire dictionary has to be searched for each combination of program name and version, which could result in a long processing time. Obviously, the processing speed of this problem could be increased with a parallel approach.

The parallelization itself should allow the full utilization of the available resources for the search in the CPE dictionary. The possible results would be either the correct CPE-ID or the absence of a CPE-ID for the given program. The first idea was to parallelize the process by dividing the CPE dictionary into equal-sized partitions. The number of partitions should be in accordance to the number of available cores. Then each combination of program name and version can be searched in multiple parts of the dictionary at the same time. If a suitable match was found in one of the partitions, the corresponding thread returns the result and all remaining searches could be stopped. Hereby, a smaller part of the dictionary has to be searched and results should be retrieved much faster. However, it turned out that the performance gain does not outweigh the additional overhead for parallelization, since the combinations for program name and version are still processed one after the other. This insight could be explained with the low processing time for a single lookup due to the small size of the dictionary. Thus, it was not beneficial to parallelize the lookup itself but the analytic approaches offer another possibility to parallelize the workload.

Therefore, another parallelization approach was implemented that utilized the fact that most analytic approaches are based on a set or a compilation of programs. The idea is to divide the set of applications, which should be processed, instead of the dictionary. Then, every thread can run independently and perform the lookup in the dictionary for its subset of name version tuples. Finally, the

results can be merged and returned for further processing. This method does not require additional synchronization beside the final collection of all intermediate results. The only requirement was that the dictionary has to be available for every thread to avoid locks and synchronization issues. In the first implementation, a copy of the dictionary was read into the memory for each thread. Since the size of the CPE dictionary is relatively small[5], the additional memory overhead could be accepted.

Figure 3.3: Parallel Lookup Performance with 4 cores



The resulting performance boost to perform the lookup for suitable CPE-IDs in parallel is visualized in Figure 3.3. The diagram shows that the speedup factor from the parallelization is bound to the number of available cores.

In a later implementation, a shared data structure was used for the CPE dictionary, as only read operations are performed. Since reading of the data structure does not have additional side effects, multiple threads can perform the reading in parallel. Thus, the requirement to reproduce the dictionary in memory was circumvented, although it might not have been necessary due to the limited size.

[5]CPE Dictionary was 41 MB on 19.02.2018

3.5 Analytic Features

This section describes analytic approaches that have been implemented in the HPI-VDB [27]. The approaches are based on the data model that was explained in Section 3.2 and illustrated in Figure 3.1. The presented approaches work in an automated fashion to allow the processing of vulnerability information. Since nowadays the amount of information increased dramatically, automated methods have to be used to handle the large number of information and possible combinations. In addition, the different approaches always aimed at a reasonable processing speed and a good usability to incorporate them into the online version of the HPI-VDB. Especially, the performance of several approaches is interesting, since the number of different applications and systems create a wide space for combinations and require powerful analytic capabilities. The requirement that all approaches should be useful for the public will reoccur and result in additional efforts, which will be mentioned later on.

3.5.1 Self-Diagnosis

The first approach that will be presented is the self-diagnosis [22]. This approach was the first analytic method that was implemented into the HPI-VDB. The idea was to provide a possibility to utilize the vulnerability information in an automated way. The procedure has to be automated, since the amount of vulnerability information increased recently. For example, more than 10,000 vulnerabilities were published in 2017, which was also the reason for the extension of the CVE-ID to more than four digits. The automated approach was designed to detect vulnerabilities on a system and report the results back to the user. The approach or part of it could also be used as a foundation for other approaches that require the initial detection of vulnerabilities. The self-diagnosis could be divided into two major steps, namely the *CPE-ID detection* and the *usage of CPE-IDs to correlate vulnerabilities*, which will be discussed in the following. First, I will explain the extraction of the CPE-IDs and its necessity. Afterwards the second step, which uses the CPE-IDs to correlate vulnerabilities and, through this, predict the existence of vulnerabilities, will be described. Later on, a short

elaboration on the requirements for implementing the approach in HPI-VDB is given and the concrete use case will be explained in Chapter 4.

CPE-ID detection

The first task in the analytic approach is to gain a deep insight to the system that should be analyzed. It is necessary to create a comprehensive representation of the systems inner structure. Then, every installed or running application is retrieved. This information allows the approach to predict the existence of vulnerabilities, since a vulnerability is mostly dependent on the affected application. Therefore, if the information about all running applications is available all the related vulnerabilities can be deduced and they are very likely to exist on the target system. The major requirement is that the application identification has to be done with a very high level of detail and a high accuracy, since many vulnerabilities only affect a specific version or a certain language pack. The requirement to identify applications with this high level of detail can be fulfilled when the CPE-ID of the application is known. As it was explained in Section 2.3.2 the CPE-ID provides a standard to precisely identify applications and their versions. Thus, the CPE-IDs are usually utilized to describe the requirements of a vulnerability. Therefore, the analytic approach can use the CPE-ID to identify the running or installed programs on the target system and to directly refer to a precise version of the application and its vulnerabilities later.

The extraction of CPE-IDs on its own is a complicated task as different programs have different installation routines. Furthermore, different operating systems use diverse mechanisms to preserve information or document installation procedures. Even the execution of a program itself could be varying. For example, one could execute an already created binary, or compile the binary and run it, or use third-party tools to download, install and run programs. These different ways of executing an application leave different traces and require suitable methods to identify running or installed programs. Thus, the self-diagnosis has to distinguish the type of operating system to use the appropriate extraction method in the beginning.

CPE-ID extraction on Windows systems One extraction method was designed to work for Windows systems. During the development, several tests with registry extraction or interpretation of environment variables, such as system path, were performed to find installed applications. However, I found a third-party tool, which already solves the problem of the program identification. The tool is the public available scanning tool OVALDI [41], which was published by the MITRE corporation and is now maintained by the Center for Internet Security. The scanner is documented on the OVAL website [8] and can be downloaded via sourceforge[6]. The scanner itself will test for properties that are specified in the definitions, which can be downloaded from the OVAL website [8]. The definitions are updated on a daily basis and are organized in various categories, such as inventory or vulnerability. So, as explained in Section 2.3.5, OVAL itself can be used to find vulnerabilities, but it only encompasses 16,000 tests for vulnerabilities, whereas the HPI-VDB contains information about 96,000, as of February 2018. This is the reason why OVAL can be used to identify applications, but when it comes to vulnerabilities, the amount of available tests lacks comprehensiveness. In addition to the different classes, the definitions are divided into groups for different operating systems and architectures, which allows working only with the necessary subset of definitions during the processing. Thus, the self-diagnosis for Windows will utilize the inventory and program identification capabilities of OVAL. One benefit when using OVAL is that the active community of OVAL continuously adds definitions and maintains the already created definitions. Another benefit of using the OVAL method is that the result of the inventory detection contains CPE-IDs for each identified application. Hence, it is not necessary to perform additional lookup operations in the CPE dictionary. Besides, the OVALDI scanner is also available for different operating systems, since it is published as a Windows executable, as well as an RPM package and source code for self-compiling.

CPE-ID extraction on UNIX systems UNIX systems could be scanned for applications with the OVALDI tool as well, since the scanner's source code is

[6]https://sourceforge.net/projects/ovaldi/

available and could be compiled on the specific Unix system. However, in several test cases, compilation problems arose and the decision to use default commands for the identification of applications was done. Usually new applications are installed as packages on most Linux distributions. The *dpkg* command can be used to install those packages, but it could also be used to identify already installed packages, since it has to keep track about the installation status as well. Thus, a small script to collect all installed applications was included to gather the required information on Unix systems. The concrete script will be described again in Section 4.3, when some concrete use cases will be described. The downside of this approach is that the results contain only the name and the version string for each application. Thus, an additional lookup is required to find the corresponding CPE-IDs for each application. This task benefits from the performance improvements that were described in Section 3.4. After the CPE-IDs for each program were identified, the self-diagnosis can continue independently from the operating system.

Using CPE-IDs to correlate Vulnerabilities

The comprehensive list of CPE-IDs for all identified applications represents the inner structure and all relevant components of the target system. This means that the vulnerability detection investigated the necessary information from the system that should be analyzed. The remaining step is to identify the related vulnerabilities for this combination of CPE-IDs. Therefore, the preconditions of the vulnerabilities will be used. As it was already explained, the most important requirement for every vulnerability is the existence of the affected application or hardware component. The self-diagnosis can already capitalize on the fact that the CPE-IDs for the affected applications of all vulnerabilities are listed in the HPI-VDB. Therefore, a query for all identified CPE-IDs will return the CPE-IDs, which are actually listed in the HPI-VDB and are related to vulnerabilities. Then those vulnerabilities can be retrieved as well. This first collection of vulnerabilities is checked for further restrictions in the preconditions, which is illustrated in Algorithm 1. As it was explained, the individual properties of the conditions are organized as members of condition-sets that are connected with a

logical operator. Thus, an evaluation of those condition-sets is required to evaluate if all necessary conditions are fulfilled and the vulnerability exists on the system. The algorithm will check if the condition-sets are evaluated successfully and return a boolean result. Therefore, it has to differentiate between *AND* and *OR* condition sets. Then the vulnerability will be appended to the set of results.

Algorithm 1 Evaluation of conditional structure of vulnerabilities

```
1: for vulnerability in vulnerabilities do
2:     programset = vulnerability.preconditions.programs
3:     vulnerabilityFound = false
4:     if programset.operator == "OR" then
5:         for cpe in programset.members do
6:             if cpe in cpeListOfTarget then
7:                 vulnerabilityFound = true
8:                 break
9:             end if
10:        end for
11:    else if programset.operator == "AND" then
12:        vulnerabilityFound = true
13:        for cpe in programset.members do
14:            if cpe not in cpeListOfTarget then
15:                vulnerabilityFound = false
16:                break
17:            end if
18:        end for
19:    end if
20:    if vulnerabilityFound then
21:        result.append(vulnerability)
22:    end if
23: end for
```

Finally an additional check for the remaining preconditions is done, depending on the type of analysis. For the self-diagnosis the *activeResourceInfluence* is tested, since it usually should be possible for a potential attacker to send input to the vulnerable application. Therefore, the application should be running and process input, which is send to it, e.g., via network or via local console access. For the self-diagnosis the remaining precondition, namely the *Range*, is

discarded, since it is assumed that the diagnosis should check for vulnerabilities in general and independently of the potential attack-range. This means that the self-diagnosis will perform a full analysis regardless of the position of the attacker. Additional analysis procedures that incorporate the attack range will be discussed in Section 3.5.3, when the attack graphs are presented.

3.5.2 Program Stack

Another special capability of the HPI-VDB is the program stack. Since the vulnerability information in the Internet is widespread and diverse, the idea arose to create the HPI-VDB in the first place. In addition, another observation was that it is not easy to always find the important information for oneself. Thus, the possibility to perform a self-diagnosis was created and multiple users took the opportunity to test their systems. However, the self-diagnosis is a short glimpse of the current state of the system, as well as the current state of the available vulnerabilities. Thus, in theory a self-diagnosis could return different results on two consecutive days, since the system might have changed or the vulnerability information might have been updated. Hence, another feature for a more continuous monitoring was required. A direct and detailed continuous monitoring of the target system is out of scope, as this would require an agent running on the system that continuously reports changes. Therefore, the new feature focuses on the continuity on the side of HPI-VDB.

The before mentioned requirements resulted in the program stack feature, which allows an ongoing reporting for selected applications. Hereunto, the applications that should be monitored are selected and stored for each user. This results in a list of CPE-IDs per user, which is called program stack in the HPI-VDB as it represents the accumulation of important programs for that user. An example program stack is illustrated in Figure 3.4, which includes two operating systems, namely Windows 8 and Suse Linux. However, the CPE-IDs of each application has to be inserted by the user manually in the first iteration of the program stack. Since the input of CPE-IDs could be complicated, an assisted method was designed that uses auto-completion after a separation of CPE-IDs

in classes of most used types, such as operating system or browser. The assisted method is especially helpful for inexperienced users and allows those users to benefit from the program stack. The disadvantage of this manual approach is that it cannot guarantee that every application of the system is represented in the program stack. This might be important for inexperienced users as it is difficult to find the important applications in the first place. The benefit is that a more experienced user can limit the number of applications, which should be monitored, so the amount of noise is kept low. A solution for this limitation is the option to directly transfer the identified CPE-IDs from the self-diagnosis, which was explained in Section 3.5.1, to the program stack. Then every CPE-ID from the target system can be continuously monitored.

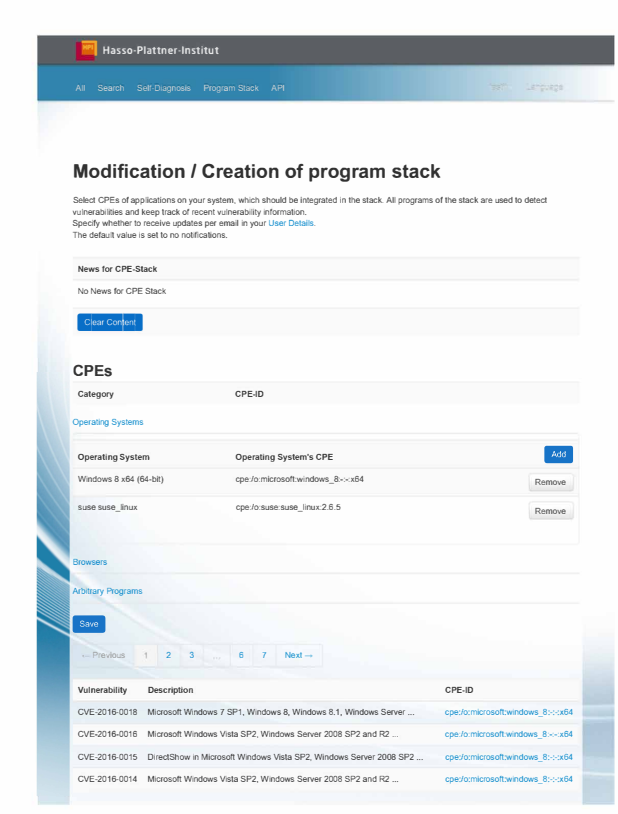


Figure 3.4: Example of a Program Stack

If the specific CPE-IDs are appended to the program stack an initial round of

collecting the associated vulnerabilities is triggered. The vulnerabilities will then be appended to the vulnerability list of the program stack. In the future, every creation of a new vulnerability that is related to a CPE-ID will consequently trigger the addition of this vulnerability to the specific program stacks, which contain the CPE-ID. This approach was realized with a signal mechanism from the Django framework[7]. Therefore, this approach helps to keep track of recent vulnerabilities for the specified applications without regular manual investigation of those applications. Furthermore, the program stack includes the possibility to receive information about updates via email. This allows users to be informed without visiting HPI-VDB regularly. The email will contain the news section of the program stack, which is updated whenever a new vulnerability was included that affects a CPE-ID from the program stack, as it was described before. The news section contains one element for each update that was detected for the program stack. The element contains the CPE-ID, the vulnerability identifier, most often the CVE-ID, and a timestamp to identify the specific update. Thus, the news section could contain several items and keep growing, as long as the user does not trigger a flush of the older elements. Currently, one third of all users actively use the program stack feature, as of 25th of February. The usage shows that the feature was widely accepted and is used to receive updates about vulnerabilities.

3.5.3 Attack Graph

This section describes the concept of the attack graph that is enriched with vulnerability information from the database. Then the attack graph could visualize the possible weaknesses in a network and reveal specific points that have to be hardened to secure the infrastructure. This attack graph differs from the traditional way of attack graphs that visualizes all possibilities as a path for one target component. This results in a large graph that is hardly readable or understandable. An example excerpt for a traditional attack graph is illustrated in Figure 3.5. This gives an idea of the amount of nodes and connections for a

[7] https://www.djangoproject.com/

relatively small scenario of five machines and three networks. In the top right corner of the Figure, the full graph is illustrated and the part, which is surrounded with the red rectangle is illustrated in the bigger part of the picture. Thus, the complexity of the traditional graph is a problem that prevents a wide utilization of the attack graph creation with MulVAL [51]. In contrast to the complex graph, the same scenario is also illustrated in Figure 3.6, which is less confusing. Several research work focused on the simplification or increase of the usability of those graphs, such as [32]. Most of the traditional attack graph creations utilize logic programming paradigms to find all possibilities that lead to a positive result. The downside is that those graphs have this high complexity. A famous example of those tools is MulVAL [51], which was also used in the example graph in Figure 3.5.

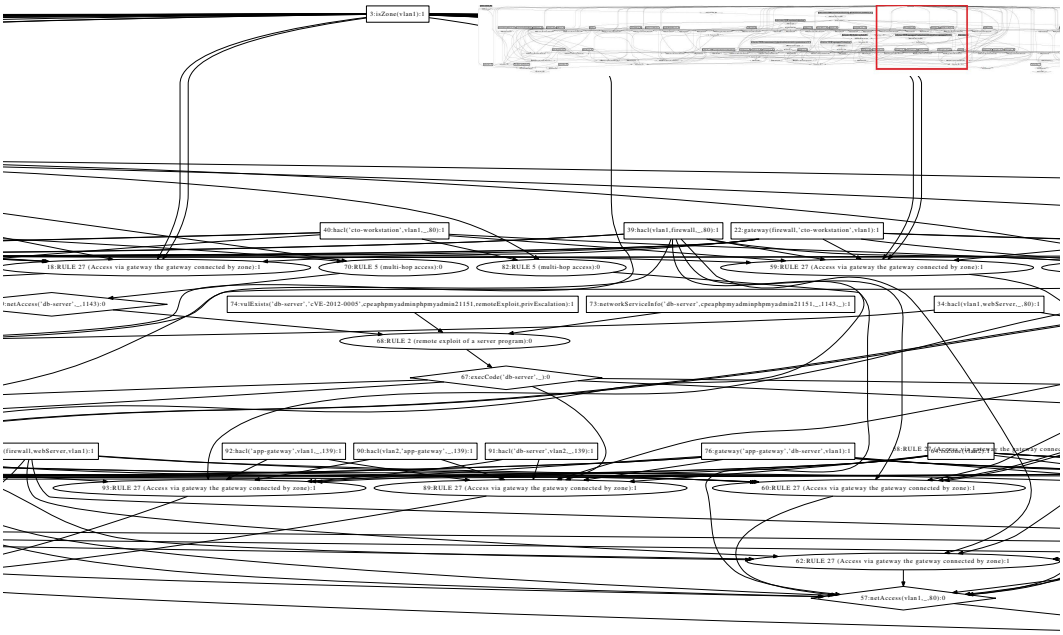


Figure 3.5: Excerpt from a traditional attack graph created with MulVAL

The first step to create this enriched attack graph is to identify the network structure and the necessary information for each machine, such as connected networks, IP addresses. That structural information will provide the foundation

for the graph creation. The different connections between the machines and their affiliations to the networks can be represented with that information. In addition, the internal state for each machine has to be detected, which means information about the running applications, installed programs and open ports have to be collected. This additional information will be used to identify weaknesses in each machine to move from host to host or elevate privileges or something similar.

The integrated network scanning tool [10] is able to fulfill both of these requirements and create a representation of the network in an XML format. This representation will contain the structural information, as well as the information about the inner structure of each node. An excerpt from an example XML representation, which was produced using the integrated scanner proposed in [10] is illustrated in Listing 3.4. If an administrator manually wants to alter this data, it is still possible and the straightforward structure of the XML allows an easy modification. The simple schema of the XML also allows the manual creation of the entire XML for smaller networks. For bigger networks the integrated scanner [10] can be used. The core requirements are that each host should have information about the connected networks and as many information about installed applications as possible. Therefore, the integrated scanner combines different tools, such as *nmap* [35] and *nessus* [63] to scan for hosts and the installed and listening applications respectively. Thus, the attack graph creation of HPI-VDB can start from the point, when a full XML description of the network is available. At this point the important applications, which listen on the individual hosts are listed in the CPE format.

So, as it was also described in the paper [23], the attack graph creation process the XML to generate the network outline first. Afterwards vulnerability information about the identified applications is gathered. This requires the CPE-IDs for the detected applications, which can be directly taken from the XML description. When the list of vulnerabilities for each node is composed, the position of the attacker is not considered. The initial attack graph only visualizes the network structure and provides insights about the existent vulnerabilities in the details of each node. Furthermore, vulnerable nodes are highlighted and the existent vulnerabilities are displayed in the graph, if the amount does not influence the

Listing 3.4: Extract of description of a web server

```
<component id="webserver">
 <desc>This is the webserver.</desc>
  <image>
   <os>
    <cpe_name>cpe:/o:microsoft:windows_xp::sp2</cpe_name>
    <port_number>139</port_number>
   </os>
   <program>
    <cpe_name>cpe:/a:apache:http_server:2.4.9</cpe_name>
    <port_number>80</port_number>
   </program>
...
```

visualization, as it is illustrated in Figure 3.6. In addition to the nodes themselves, their connections are highlighted as well, since connections that originate from a vulnerable host are classified as corrupted as well, if the vulnerability affects the *integrity* of the system. In this case, the attacker could modify data and execute commands on the system, which results in an overall compromise of the system. Then, the network connections from this system are under the control of the attacker as well and cannot be trusted anymore. Thus, these compromised connections visualize the possibilities of an attacker to influence the data flow in a network and move from one node to the other. In addition to the general evaluation of the network, it is also possible to explicitly place the attacker in one of networks. Consequently, an evaluation is triggered that computes the hosts, which can be accessed or captured by the attacker. Therefore, it is a similar evaluation to the approach already described but this time the location of the attacker is considered as well. The data structure for the graph is a JSON representation of the individual nodes and edges with their corresponding attributes. Then the visualization of the graph is performed by Springy [30], which is an external library and able to create a force directed graph. This graph should have a minimum of overlapping edges and the nodes should be distributed equally in the available canvas. Thus, these properties should result in a well-structured graph that does not include additional complexity, as it is

shown in Figure 3.6. The graph itself allows interaction with the nodes, so one can easily drag or remove individual nodes. For the presented graph, the Springy library was extended to incorporate the possibility to highlight edges and nodes and visualize the vulnerability identifiers, as it was described before. Finally, the graph is visualized using the extended Springy library and JavaScript in the browser. One constraint of the simplistic graph is the limited number of nodes that still allow a good representation and the interaction of the user. If the target network contains tens or hundreds of nodes, the approach does not perform well and another library should be used to visualize the graph. However, the general approach and the reasoning can still be utilized, only the visualization front-end should be replaced.

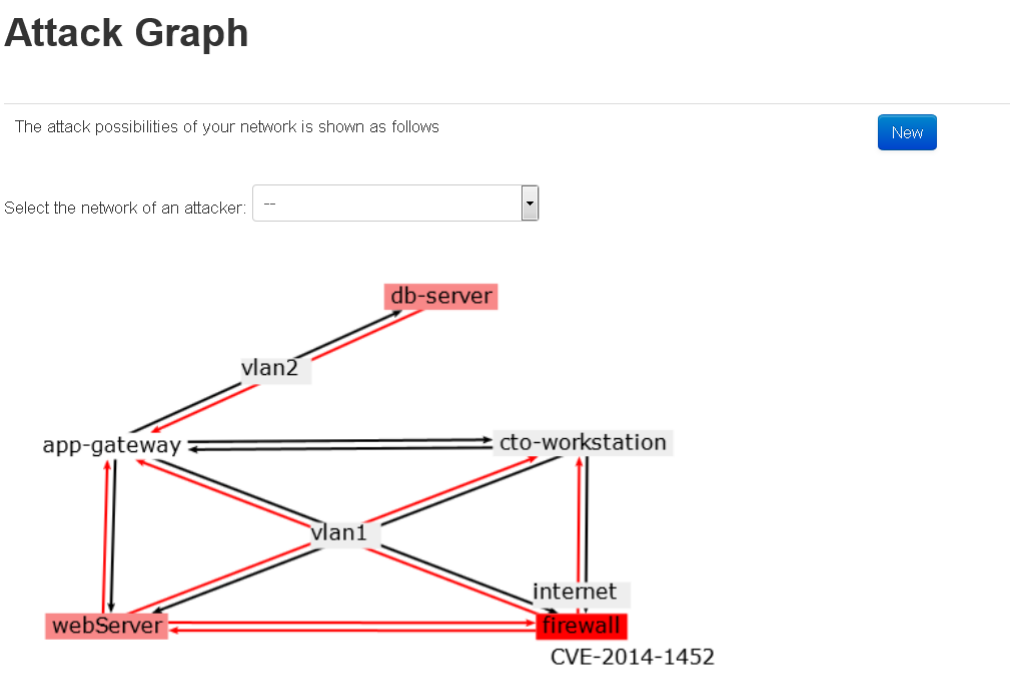


Figure 3.6: Example Attack Graph without an attacker

Overall, the graph from the HPI-VDB, which is shown in Figure 3.6, increased the readability of attack graphs, but the downside is that the attack routes are not

visible. This attack graph only visualizes the network structure and integrates information about the vulnerabilities and endangered nodes of the network. Nevertheless, the HPI-VDB can still be used, as a knowledge base for vulnerability information, with the original attack graph creation of MulVAL as it was explained before and shown in Figure 3.5. Therefore, the information could still be produced, but the HPI-VDB incorporates the more simplistic representation of the attack graph as the detailed representation was designed for a special group of experts only.

Furthermore, the solutions for the vulnerabilities were integrated as well. Therefore, the solutions from each vulnerability are collected and duplicates in the resulting set of solutions are dismissed. Then, a list of available solutions is presented to the user, which can be applied by selecting the solution. The result of applying the solution to the scenario will be represented in the attack graph. This means that the attack graph is updated interactively and could also be used to evaluate the effectiveness of the possible solutions. Based on this, decisions could be made, which solutions or which fixes have to be prioritized, because they have the largest impact on the security of the network. It can also be used to identify focal points that should be secured especially, since most attacks have to go through this station.

3.6 Conclusion

This chapter describes the fundamental model that was used to represent vulnerabilities with all necessary properties to allow the application of analytic approaches. The unique attributes of the proposed vulnerability model are the preconditions and the postconditions. Those conditions aim at the overall goal of the HPI-VDB [27] to allow an automatic processing of the vulnerability information. During the development and the research, other vulnerability databases, such as NVD [46] also integrated requirements for a vulnerability which are similar to the preconditions. However especially the postconditions allow an evaluation of possible combinations of vulnerabilities, which could be exploited one after each other.

Thus, the proposed attack graph was an obvious choice for one of the implemented analytic approaches. The simplified attack graph differentiates from traditional attack graphs, which tend to be confusing, because of their complexity. The presented attack graph illustrates the network structure, which represents the inevitable complexity to model the target network, with integrated vulnerability information.

Beside the attack graph, the vulnerability detection is the major analytic approach that can be used to scan a system and identify existing vulnerabilities. The detection relies on the before-mentioned preconditions and automatically evaluates the necessary requirements for each vulnerability. This approach was also developed with a high focus on performance improvement, as it has to be applied to each system of a target network, which could require numerous computations. Furthermore, the results of the vulnerability detection could also be incorporated into the program stack feature. This feature allows to receive information about recently discovered vulnerabilities for the specified applications. Thus, the users do not have to constantly check for information themselves anymore. They merely have to specify the applications of interest and receive notifications about updates automatically. Several users benefit from this feature already, which shows the acceptance and usefulness.

A special requirement during the development of the HPI-VDB and its analytic approaches originates from the fact that the system is publicly available. This means that the ability to demonstrate the approaches to the public should always be considered. Additionally, each approach should be understandable for novice users and it should have an intuitive workflow. Besides, the requirement to make the approaches usable via the web server also results in performance optimization as the average user does not want to wait for a request for a long time duration.

Chapter 4

Use Cases of Vulnerability Information

4.1 Introduction

The original purpose for the development of the HPI-VDB [27] was to create a database for vulnerability information that can be automatically evaluated and queried by analytic approaches. Since most other publicly available vulnerability databases limit the access to the data or even do not provide the possibility to query the data in a large scale, the necessity to create an own database had to be satisfied. Additionally, the format of the available vulnerability information is often designed to be readable by humans and it is hard to extract the most relevant information for the analytic approaches. Thus, the HPI-VDB was created and provides the data in a machine-readable format. Furthermore, it allows querying the database frequently for specific characteristics of the vulnerabilities and the results will comply with the desired format for different analytic approaches. The approaches were partly integrated into the database itself to provide additional benefits to the users of the HPI-VDB. This chapter will elaborate on some approaches that were developed for different use cases for vulnerability information.

4.2 Attack Graphs

The first use case for vulnerability information is the creation of attack graphs. The attack graphs utilize vulnerability information to detect possible paths of an attacker in the network. Therefore, the network structure has to be identified in the first place. As it was described in Section 3.5.3, one could use different network scanners, such as nmap [35], or even vulnerability scanners, such as nessus [63] to create the initial outline of the network. Another possibility would be the usage of the integrated scanner, which was described in [10]. The integrated scanner has the benefit that the result of the network scan can be directly used in the attack graph creation process. As it was described earlier, the scanner produces an XML file, which will be parsed to identify the overall network structure and running applications. Since the structure of the XML is kept in a simple form, administrators can manually adjust the XML file, if the scanning was inaccurate. Then two different possibilities to create attack graphs have been created. One attack graph creation was already described in Section 3.5.3. This attack graph is directly included in the HPI-VDB and provides an overall picture of the weaknesses in the network. It includes the possibility to place the attack at various points in the network structure and use the reasoning to identify the possible paths. If no attacker is placed, a complete evaluation of all connections is executed that highlights suspicious connections and herewith disclose possible attack paths, which could be used by an attacker. The attack graph is visualized interactively, which allows the user to modify the graph, by dragging or removing vertices. Thus, a complex graph could be modified to investigate a specific part of the attack graph. Furthermore, the vulnerability information that has been used to identify affected nodes in the network, are attached in a textual format to the nodes. This allows an easy investigation to identify the vulnerabilities for each node.

As it is illustrated in Figure 4.1, an additional table illustrates the vulnerabilities per node and includes a preview of the textual description for each vulnerability. Besides this preview, the other valuable information that is included in the overview is the affected software. The affected application is the

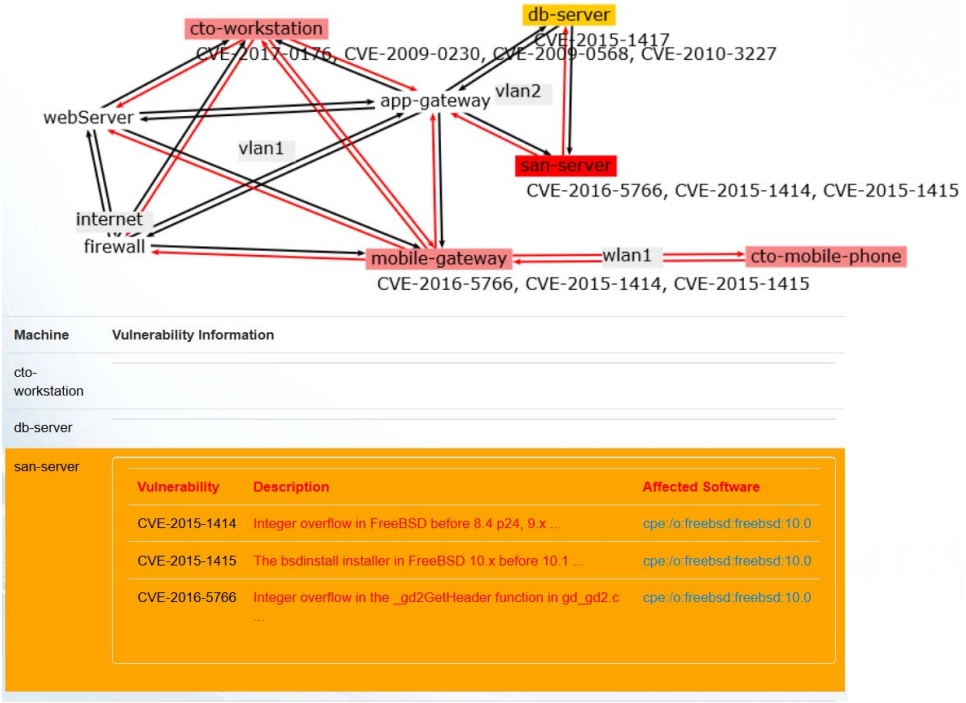
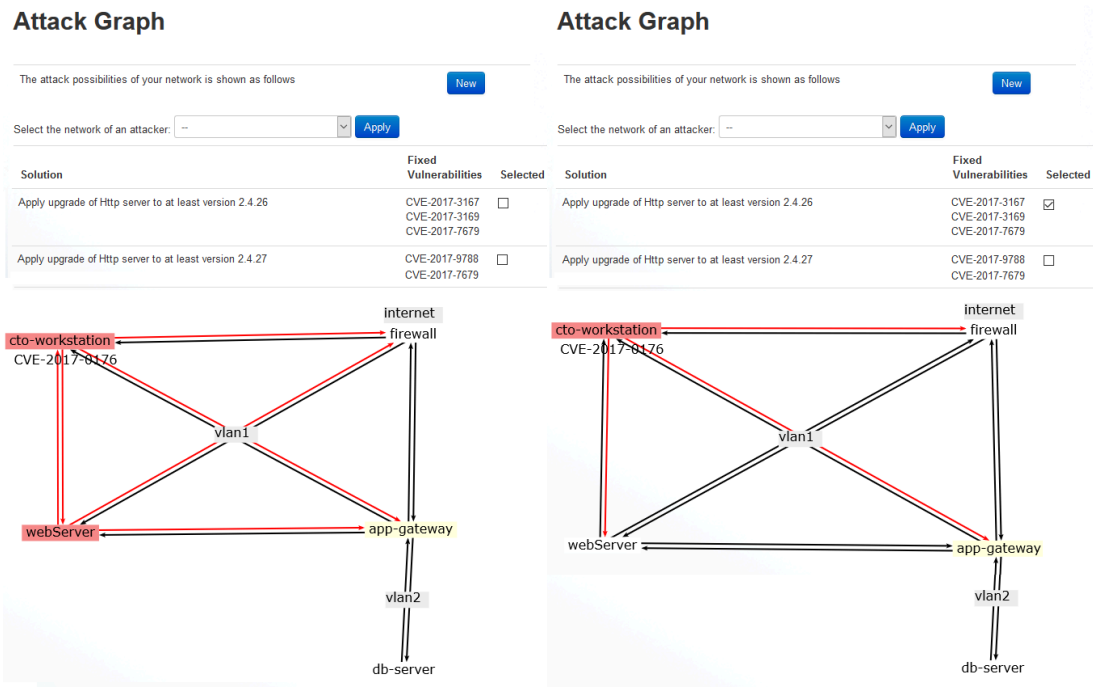


Figure 4.1: Attack Graph Visualization with Vulnerability Information

original cause for the existence of the vulnerability and therefore security experts will most likely use this information to resolve the vulnerability through an upgrade or an uninstallation. The visualization also includes information about the type of vulnerability. The “db-server” for example contains one vulnerability, which does not infect the integrity of the node. Thus, the node is marked as orange, since it contains a vulnerability that can lead to data leakage, but the vulnerability will not allow a compromise of the system. The endangered systems that can be compromised and abused to penetrate deeper into the network are on the other hand marked as red. This color-coding should allow a fast recognition of important and high risks, which should be the focus for the security experts. An additional feature for security experts is the integration of solutions into the attack graph view. Above the graph itself, the available solutions for

each vulnerability are listed and combined if a single solution resolves multiple vulnerabilities. Then, the security experts can manually select solutions and the graph will be recreated with the selected solutions applied. Thus, it is possible to identify the minimal effort to resolve all relevant vulnerabilities and secure the network again. The combination of solutions is necessary as it could occur that an old version of some application contains a vulnerability, which was fixed in later version. Then this later version could again contain a vulnerability, which was fixed in the current version. This would mean that it is not necessary to apply an upgrade to the intermediate version of the application to resolve the first vulnerability, since this vulnerability will be resolved with the current version of the application as well. Therefore, only one upgrade to the latest version is necessary. An example of this attack graph with integrated information about solutions is illustrated in Figure 4.2.



(a) Without Solutions (b) With Solutions

Figure 4.2: Apply Solutions for Attack Graph

On the left side, the default attack graph is illustrated in Figure 4.2a. It visu-

alizes an attack graph of a network that contains five hosts with two individual networks. The *vlan1* network contains a *cto-workstation*, a *firewall*, a *webServer*, and an *app-gateway*. The *app-gateway* regulates the access to the second network that includes the gateway and a *db-server*. Furthermore, the *firewall* secures the connection to the internet. The Figure also highlights the two vulnerable hosts of the scenario, namely *cto-workstation* and *webServer*. The *webServer* contains an old version of the Apache HTTP server, with several vulnerabilities. These vulnerabilities could be fixed by an upgrade of the HTTP server, as it is indicated in the solution table of Figure 4.2a. On the right side, the result of the application of the solution to the target scenario is illustrated. The security expert checked the upgrade of the HTTP server to version 2.4.26, which resolves three vulnerabilities. Then the *webServer* is not vulnerable anymore and the attack graph adapts itself accordingly.

The second attack graph creation was developed in an external attack graph construction platform (AttGCP)[9], which uses the Multi-host, Multi-stage Vulnerability Analysis Language (MulVAL) [51]. The requirements for the attack graph construction platform were originally a system information file, which should have the same format as the XML file that was used in the other attack graph approach and another XML that contains the knowledge base for vulnerability information. The development for the platform included the possibility to retrieve the vulnerability information directly from the HPI-VDB server. Thus, the vulnerability information should always be up to date and the platform only needs to process the relevant vulnerabilities, because the HPI-VDB can already return a filtered set of vulnerabilities. Beside this improvement, the reasoning that is used to identify possible attack paths was enhanced, because some logical errors have been identified. The platform uses the MulVAL reasoner, which is written in the logic programming language Prolog [68]. The requirements for the reasoning with Prolog are a knowledge base that describes some facts and possible relations and rules between these facts. During the execution, the Prolog engine generates all possible variable bindings and tests whether this combination results in a true statement. If a true result is the logical consequence, the used variable binding is returned as a possible solution. If a binding results in

a negative result, the bindings after the latest choice-point are newly set with the next possibility. This process is also called backtracking. Thus, it iterates through all possible variable bindings and finds the binding that leads to a positive result. This behavior is used in the area of attack graphs, where a goal is specified beforehand. The usual goal is the access to a server, a file, or a successful denial-of-service attack. Then the starting point for the attacker has to be specified, which is often the location of the attacker. Additionally, the network structure has to be represented with facts, which is necessary since the network structure includes fixed environment factors and the overall setting of the scenario. Now, the MulVAL interpreter can start to iterate through all possible combinations of rules and relations with the different variable bindings. If all possibilities have been tested, the combinations with a positive result represent possible attack paths through the network. Finally, those paths are combined for the attack graph, which is also the reason for the complexity of those graphs, as it is illustrated in Figure 3.5.

4.3 Detection of Vulnerabilities

The most important use case for vulnerability information is the detection of those vulnerabilities in specific systems. Besides the human factor, another huge impact factor is the existence of vulnerabilities for successful attacks. The detection of vulnerabilities represents the fundamental needs for vulnerability databases in general. Therefore, the vulnerability information has to be publicly available. Then, security experts could utilize this information to ensure that their systems are not affected by any of the published vulnerabilities. However, this traditional approach was often cumbersome and required a tremendous manual effort. Especially, if the systems become more complex and contain numerous applications and components, the manual maintenance is hardly possible anymore. Thus, the mechanism to process, understand, and utilize vulnerability information to identify vulnerabilities on specific systems have to be automated. Hence, the HPI-VDB was created with the special requirement to include machine-readable information for each vulnerability. These information should

then be used in automated detection approaches. As it was already explained in Section 3.5.1, preconditions and postconditions are assigned to each vulnerability during the import phase. Those conditions are available in an XML format and represent the requirements and the impact of a vulnerability. For the detection, the requirements of a vulnerability are the most important characteristics, since each of the requirements has to be satisfied to exploit the vulnerability. Furthermore, the most important part of the preconditions is the existence of the affected application or if multiple applications are affected, at least one of them has to be installed. Therefore, the main feature in the process of vulnerability detection is the comprehensive identification of applications and additional details of the system. At the point when the interior state of a system can be represented with all installed, running, and listening applications and additional configurations, the real vulnerability analysis can start. The applications have to be detected with a high level of detail to allow specific identification of the version or additional release differences, such as language packs or architecture. Therefore, the official standard CPE is used and each application is associated with the corresponding CPE-ID. Afterwards, the complete list of detected CPE-IDs is used to find all vulnerabilities that are related to at least one CPE-ID, as it was also explained in Section 3.5.1. Finally, the resulting vulnerabilities are presented to the user in a list that is sorted by the severity of the vulnerabilities, as it was also described in [22]. Thus, the most important and most critical vulnerabilities are highlighted and could be investigated for possible solutions.

As it was described before, the overall process of vulnerability detection heavily relies on the correct and comprehensive detection of the applications on a target system. Thus, this step is the most important part of the approach, since the following correlation in the database can be easily achieved. However, a complete investigation of a systems interior poses a more challenging task. Therefore, the OVALDI tool [41] was used for Windows systems and even though it also exists for UNIX systems a small script, which is illustrated in Listing 4.1, was the better solution.

```
#! /bin/sh
echo "This script will collect the names of all installed
   applications\n Continue? (y/n)"
while [ "$Key" != "y" ] && [ "$Key" != "n" ]
do
read Key
if [ "$Key" = "y" ]
then
echo "Creating inst_pkgs file"
dpkg -l | grep -Po "^ii\s*\K\S+\s+[0-9\.:]+\d"| sed -E 's
   /[0-9]://g' > inst_pkgs
elif [ "$Key" = "n" ]
then echo "Abort"
fi
done
```

Listing 4.1: Script to gather application names and versions from Unix Systems

The script includes a short information for the user and requires the explicit confirmation to be executed and lists all packages of the systems and filters them by the installation status. Then it selects the name and the version of each application. Finally, the list of applications is transmitted to the HPI-VDB, which has to translate the combinations of name and version to the corresponding CPE-ID, before the workflow is similar to the detection on Windows systems again. The script was preferred to the OVALDI solution, since several tests indicate runtime and compiling problems for OVALDI on UNIX systems.

However, an additional challenge has to be solved to provide this feature to average users. The usability for the script is already satisfying since it can be downloaded and easily executed. One advantage of the script is that it does not require superuser privileges and the limited complexity even allows a manual inspection of the commands that should be executed. This should increase the trustworthiness of the approach, which is a special requirement for the vulnerability detection and will be discussed later. Furthermore, the user is presented with a set of instructions that describe the workflow for the vulnerability detec-

tion on UNIX systems. After the user downloaded the script, he should execute it and upload the resulting file to the vulnerability database. Then, the correlation process will be started and the detected vulnerabilities are reported to the user.

The usability of the Windows based approach had to be improved as well, since the OVALDI scanner can be downloaded and executed without installation, but it requires the correct definitions and has to be executed from the command line. Those two requirements could usually be difficult for average users, since they require advanced knowledge. Thus, the vulnerability detection was packed into one executable that contains the current version of the OVALDI scanner. It automatically downloads the latest and the correct version of the definitions that are needed by the OVALDI scanner. As it was described earlier, the OVALDI scanner relies on a knowledge base that is provided on the OVAL website and maintained by the OVAL community. However, this results in the necessity that the definitions have to be downloaded before the analysis can start. The repository contains definitions for vulnerabilities, compliance, patches, and inventory. Since the vulnerability definitions always lack some recent and some historic vulnerabilities the executable will download the inventory definitions. Thus, it is sufficient to circumscribe the OVALDI detection to the identification of all installed applications. Therefore, the inventory definitions include tests to detect all installed applications with their version numbers and additional details. Furthermore, the output of the application detection contains the CPE-IDs of the detected applications. Hence, it is sufficient to perform the OVALDI inventory detection to extract all CPE-IDs from the target system. Finally, the bundled executable establishes a connection to the HPI-VDB and performs a login to the desired user. Then it uploads the resulting list of CPE-IDs in the user's private area, which is then used to perform the vulnerability analysis, as it was described in the UNIX based approach.

4.3.1 Requirements and Limitations

Since information about the interior of a target system is sensitive data, the approach includes additional protection measures to guarantee the privacy and the confidentiality of the data. This requirement arises, because the situation can become dangerous if an attacker is able to receive this information about a possible victim. The application detection results include all installed applications that were identified on the system. Either this information can be used to identify vulnerabilities and reveal attack possibilities on the victim's system or it can also be misused to perform a social engineering attack. A social engineering attack might be successful through pretending to be a support for one of the installed application and try to deceive the victim to reveal further information or obtain direct access to the victim's system.

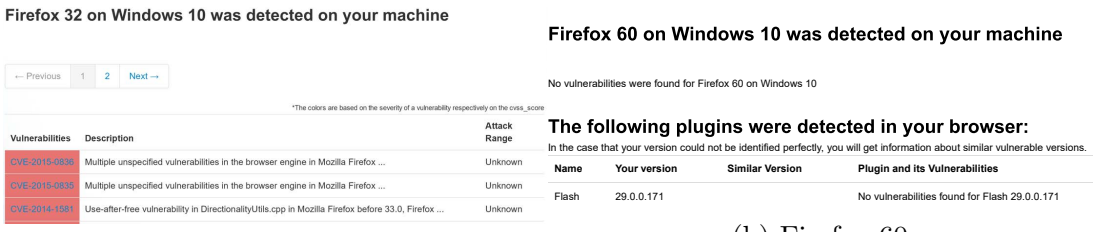
In addition, recent incidents also increased the user awareness for personal and sensitive data, which would lead to an aversion of using the service without additional security mechanisms. This is also the reason why every approach is created in a transparent way to allow users to comprehend every single action if they chose to. Nevertheless, even with the previously described methods to ensure confidentiality and privacy of the user's data, the operation of the HPI-VDB still shows that many users are scared to reveal so many details about their systems. The full self-diagnosis was only used from 10% of the registered users. The approach requires a user to be registered, since the list of CPE-IDs or program name and version combinations has to be uploaded to the HPI-VDB. Then the uploaded information has to be stored and protected to only be available to the one user who performed the upload. Even though the details about the user's systems are protected with the highest security mechanisms, the observation that many users do not perform the full vulnerability detection still holds true. Thus, another detection method had to be introduced into the HPI-VDB to make the service beneficial for average users. The most important requirement is to only depend on a minimal set of information from the users. Ideally, the approach should only require information, which is already available if the user visits the website. Thus, the idea to have a browser based vulnerability

detection was created, which will be described in the following.

4.3.2 Browser Based Vulnerability Detection

The browser based vulnerability detection can be executed without the requirement of additional information from the target system beside the information that is already available. The detection utilizes the fact that the target system connects to the web server with a normal HTTPS request. This request already contains meta information in the HTTP headers, such as *accept-language*, *host*, *referrer*, or *user agent*. The most important header is the *user agent*, since it provides information about the browser and the system that performed the request. Usually, this information is used for content negotiation, which means that the server adjusts the source of the website to consider browser specific differences. These differences include some HTML tags or cascading style sheet(CSS) items, such as the different prefixes of CSS values ("-moz-", "-webkit-", "-ms-" for mozilla, chrome, and microsoft respectively) or *xml*, and *comment* tags. In the case of vulnerability detection, the user agent can be used to identify the browser of the client, including its version and it can be used to reveal details about the operating system of the user. So, a normal user agent could look like: "$Mozilla/5.0$ ($Windows\ NT$ 10.0; $Win64$; $x64$; $rv:32.0$) $Gecko/20100101$ $Firefox/32.0$". This user agent reveals that the client is running Windows 10 as an operating system and it is using a 64-bit architecture. Furthermore, the browser is a Firefox of version 32.0. Thus, the vulnerability detection can focus on this information to identify vulnerabilities on the target system. The browser is often considered as the main intrusion point and should be especially secured and protected. Additionally, on normal client systems the browser is the only application that establishes connections to the network and could be abused by attackers to infiltrate the client system. On top of this, the information about the browser is send on every request, which means that the attacker is also able to derive the browser and identify possible vulnerabilities. Therefore, the detection is finally able to identify the browser and all related vulnerabilities are returned. The second part of the information that refers to the

operating system cannot be fully used, since it does not provide enough details. The user agent only allows an identification of the major version of the operating system, such as Windows 7, Windows 8, Windows 10, or Windows XP. Unfortunately, it does not include further information as the service pack or patch levels. Thus, the available information from the operating is not detailed enough to make a precise assessment and derive the related vulnerabilities. However, the information is used to further filter the resulting vulnerabilities from the browser. The idea is to remove all vulnerabilities that require different operating systems. Therefore, if the example user agent is considered all vulnerabilities that require a Unix system are removed from the resulting list of vulnerabilities. In this example the user would perform the browser based vulnerability detection and he would receive a result that is illustrated in Figure 4.3a.



(a) Firefox 32

(b) Firefox 60

Figure 4.3: Browser Detection

Afterwards, a normal user behavior would be to perform an update of the browser to the actual version at this time, which is Firefox of version 60, to resolve the detected vulnerabilities. Thus, in the second vulnerability detection the updated browser was detected without any vulnerabilities, as it is illustrated in Figure 4.3b. In the result, no vulnerabilities could be associated, because at this time no vulnerabilities have been known for this browser version. However, the HTTP header that contains the user agent could be forged and there are some plugins that allow a manual specification of the user agent. In these cases, the browser based vulnerability detection will not work correctly, but since the user performs this concealment method himself, he should be aware of the implications by a forged user agent. Another limiting factor is the accuracy of the browser version in different browser families. Firefox browsers include the version with

the major version number and one additional digit and chrome browser even include the full version string, whereas internet explorer and edge only include the major version number. These differences have to be considered as well, which means that it is not possible to detect the detailed version for edge and internet explorer. Furthermore, it was observed that the ESR releases do not include a special indicator in the user agent, which results in the appearance of an outdated browser. Beside these limitations, it is still important to prioritize the security of the browser and utilize the already available information in the vulnerability detection. This service is also the most used part of the HPI-VDB and receives between 200 and 600 requests per day on average.

4.4 Passive Vulnerability Detection

The machine-readable vulnerability information can also be used in another analytic approach. Since, users tend to hesitate with releasing sensitive data, this approach will utilize already available information. In these terms it is similar to the browser based detection. However, the passive vulnerability detection is designed to work in corporate networks of companies or organizations, which was also described in [25]. As it can be observed in the previously described approaches, the most important challenge in detecting vulnerabilities on target systems is to correctly identify the installed applications. The remaining correlation between the identified applications and the corresponding vulnerabilities can be achieved with a suitable data structure, such as the vulnerability model, which was described in Section 3.2. Then the correlation can either be performed by manual investigation of the detected applications or with the implemented application programming interface(API), which will be explained in Section 4.5. Thus, the focus for the passive vulnerability detection is the identification of installed applications. As the name suggests, passive methods are utilized to perform this identification, which should have a lower impact on the performance compared to active periodic scanning of the target systems. Similar to the previous approaches the results of the application detection should have a high level of detail and include the CPE-ID for each detected application. Then

the correlation of vulnerabilities to each CPE-ID can be directly reused from the other approaches. The identification of the CPE-IDs requires deep insights into the target systems. Usually, active scanning methods, such as Nessus [63] or OVALDI [41] can be used to extract all relevant details. However, those tools use multiple scans or numerous plugins to verify and increase the accuracy of the result.Compared to that, the passive vulnerability detection benefits from knowledge about the history of the system or the network, which is recorded in log files and provides a high level of detail.

An additional assumption to realize this approach for corporate networks is that those networks are constantly monitored and the occurring events are logged. This monitoring is already common practice in multiple large productive networks, because companies want to be able to trace suspicious or malicious activity. Thus, these logs contain a lot of valuable information for the vulnerability detection already. Usually, the log files are either stored and generated at a central instance, such as a domain controller or a security information and event management system (SIEM) of the network, or they can be stored at each individual host first and forwarded for later analysis. If the logs have to be forwarded, existing tools can be used to transfer the information. Some of the more popular tools are (r)syslog, which uses the syslog protocol [47], or Windows Remote Management(winrm) [39], which can forward Windows Event Logs. Hence, it is always possible to gather the information at a central point of our network and it is not necessary to rely on agents that have to be pre-installed on each client in the network. This property allows the passive vulnerability detection to be directly integrated into network based intrusion detection systems (NIDS) or into SIEM systems. An example of a SIEM system that could be used to integrate the passive vulnerability detection is the Real-time Event Analysis and Monitoring System (REAMS) [28][1], which was developed in our research group. The SIEM systems usually also perform a normalization of the gathered event logs. The normalization is necessary to process log formats of different applications, which are often unique for the application. Therefore, the application is derived from the log message and afterwards an application specific regular expression is used to extract the desired information from the log line. In our REAMS the

normalization is performed hierarchically, which can be directly mapped to the hierarchical object log format [55]. Since this feature requires the preliminary identification of the application that produced the log, the information about the application is already available in the SIEM system and can be used in the vulnerability detection. Furthermore, this feature allows an easy extension and inclusion of new clients that should be integrated into the vulnerability detection. The new clients only need to forward their log files to the central instance as well. An advantage of the integration into NIDS or SIEM systems is the ability to correlate occurring events with vulnerability information. Since the preconditions of the vulnerabilities can be interpreted, it is possible to evaluate which preconditions are already fulfilled and keep track of the state of the missing preconditions. So, the progress in a possible exploitation of vulnerabilities can be lively monitored as well. Since the crucial part of the passive vulnerability detection is the identification of installed applications, two different approaches that utilize log files will be presented in the following.

4.4.1 CPE Detection based on System Logs

The first approach to identify installed application utilizes information from log files that are automatically created by the operating system or exist on the target system itself. The level of detail in those log files depends on the individual configuration. However, most applications provide details about their name and their version during their startup phase or during the installation process. The advantage in processing log files is the predefined format of the information, which simplifies the information extraction process. The goal of this approach is the extraction of the application name and its version, which means that the system logs contain a sufficient level of detail about all installed application. Nevertheless, different operating systems handle the installation routines in different ways, which is the reason for a differentiation between UNIX and Windows systems.

Unix systems usually create application specific logs in the default logging directory (*/var/log*). Nevertheless, it is possible that application differs in this

behavior and write their logs to custom locations. Then running applications will create entries in their log files and document ongoing actions depending on the log level or additional configuration. However, the system often stores information and logs of events in the *syslog* file, which includes the date, the running process, and detailed information about the current event [47]. The type and the format of the logged information can be adjusted in the syslog configuration file. Additionally, useful information for this use case can be extracted from log files of the package management applications, such as *dpkg* or *apt-get*. These commands are normally used to install new applications on UNIX system, which results in the fact that those installation methods leave traces in the logs of the package management applications. The major advantage in using the logs of package management compared to the logs of the individual applications is the fixed format of the package management applications. The individual applications might have their proprietary format in the logs, which could complicate the information extraction. However, by using the information from the package management this problem can be circumvented. This information contains details about the application that have been installed with the package management, including the application name and the version that was installed. The package management has to keep track about the installation status of all applications and their versions to be able to verify if a concrete version is installed or if a new version is available already. Thus, this information can be extracted with a regular expression that captures the *program* and the *version* in named groups for each entry if the information is preceded by the word *installed*. The final regular expression looks like "$.*?installed\s(?<program>\S+)\s+(?<version> \S+)$". The fact that neither the program name nor the version contains any white-space characters allows the expression to identify the program name and the version with white-space characters as the delimiters in between. An example of the log messages of *dpkg* is illustrated in Listing 4.2, which contains information about the installation of *tcpdump*. If the regular expression is applied on these logs the second line will match the expression and the named groups *program* and *version* will contain *tcpdump* and *4.2.1-1ubuntu2.1* respectively. Since the inner state is not only changed by installation but also by uninstallation of applications, these

actions have to be identified as well. Thus, a similar expression can be used to identify the uninstallation of applications to keep track of the current interior status of the target system.

Listing 4.2: Sample Loglines of dpkg log

```
2017-12-05 06:34:45 configure tcpdump 4.2.1-1ubuntu2.1 <none>
2017-12-05 06:34:46 status installed tcpdump 4.2.1-1ubuntu2.1
```

Finally, all of those logs on the individual systems can be forwarded with the rsyslog tool that was mentioned earlier. Then, the information can be collected at the central point of the network and the application detection can be applied at this central system only. So, it is not necessary to deploy the application detection on each individual host, but solely on the central instance, which can map the information back to the individual clients. Besides, it is also valuable to collect the information about installation processes in the SIEM or the NIDS.

On the other hand, Windows systems produce information about installed applications as well. The level of detail depends on the predefined policy level of the Windows logs. However, it is possible to gather information about the installed applications based on these logs. A shortened sample of an event, which was produced during an installation routine is illustrated in Listing 4.3. The interesting information is stored in the *EventData* field and the fixed structure of the events in XML, allows a direct access to this field by using the field hierarchy. It contains the name and the version of the application that was installed on the target system. The origin of this event is the MSI-Installer process that is usually used to install applications on Windows machines.

Listing 4.3: Windows Event

```
- <Event xmlns="http://schemas.microsoft.com/win/2004/08/
    events/event">
- <System>
  <Provider Name="MsiInstaller" />
  ...
  </System>
- <EventData>
```

```
<Data>Adobe Reader XI - Deutsch</Data>
<Data>11.0.00</Data>
...
</EventData>
</Event>
```

Furthermore, the Windows registry contains valuable information about installed application that could also be extracted. Finally, the logs could also be forwarded using the winrm tool [39]. This allows a processing at a central instance and removes the necessity to deploy the processing at each client in the network. The clients only need to be configured to forward the logs appropriately.

In the end, the gathered information from both system types about the installed application contain the application names and the version. Thus, the remaining step to derive CPE-IDs is to perform a lookup in the CPE dictionary, which was also described in the performance improvement part in Section 3.4.

4.4.2 Proxy Logs and Web Server Logs to identify Applications

As it was already mentioned, the detection of vulnerabilities is especially crucial for applications that perform or allow network connections. It is important, since they are directly connected with the internet and thereby usually represent entry-points for the attacker. For corporate networks, there are usually different systems that are used by the clients of the network, which then can be used to collect information about all clients at once. One of those systems could be an internal web server and another one could be a proxy server that is used to establish outgoing connections. The web server is a good example, because it is designed to allow connections to the server itself and provide different resources. The web server produces logs, which could be used to extract the details for the web server application, but even more important is the information about all clients that connect to the web server. The logs contain information about the browser of the clients that is the most crucial application for a normal client, since it is generally used to connect to the network. A sample log of the web server

is illustrated in Listing 4.4, which documents the access from Firefox browser of version 34. The client also used a 64-bit version of Microsoft Windows 8.1.

Listing 4.4: Sample Logline of Web Access Log

```
123.456.789.211 - - [07/Feb/2016:07:37:02 +0100] "GET /vulndb/
details/CVE-2007-1192 HTTP/1.0" 200 1146 "https://hpi-vdb.de/"
"Mozilla/5.0 (Windows NT6.3; WOW64; rv:34.0) Gecko/20100101
Firefox/34.0"
```

As it was described in Section 4.3.2 the focus of the vulnerability detection is the browser, because the information about the operating system lacks sufficient details. However, this limitation does not hinder this approach, since the accurate detection of the browser is even more important. The experiments, which should prove the feasibility of the approach, focused on the two most important browsers, which are illustrated in Table 4.1.

Web Browser	Market Share
Google Chrome	68.0 %
Firefox	19.1 %
Internet Explorer	6.3 %
Safari	3.7 %
Opera	1.5 %

Table 4.1: Market Share of Web Browsers in December 2015 [67]

For those two browsers, namely Firefox and Chrome, the major release dates were mapped to the recorded events of the web server of the HPI-VDB. This results in an insight about the update behavior of the clients, which is important if a corporate environment is considered. In a company network, it is essential that all clients have a high security awareness, since one client with a lower awareness could serve as an initial intrusion point for attackers. Therefore, it is important that the clients of a company network are aware of existing updates and install them to mitigate potential vulnerabilities. Nowadays, the update behavior for browsers is different from most other applications, since browsers usually perform the updates automatically. These automatic updates should

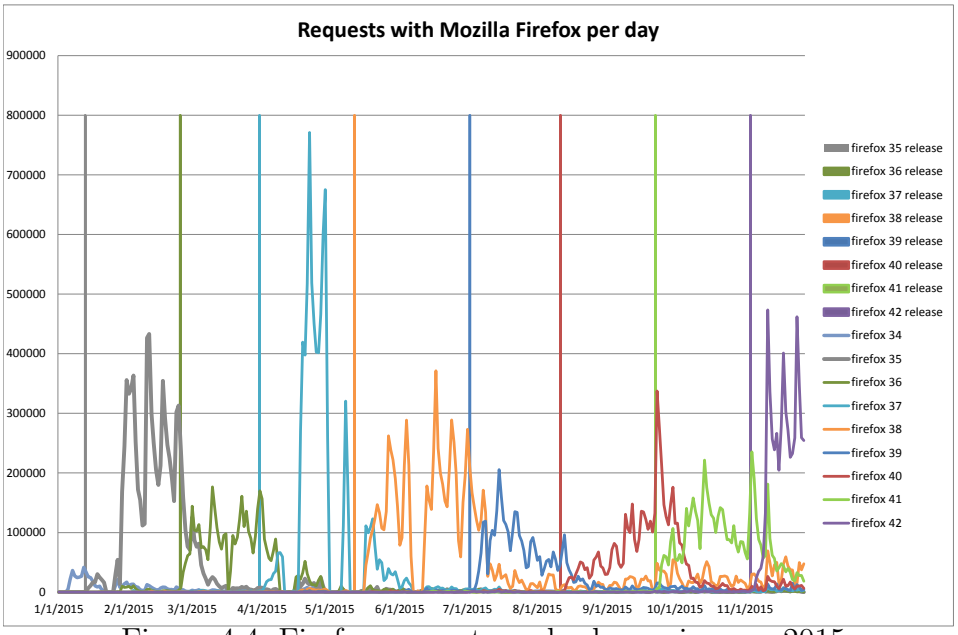


Figure 4.4: Firefox requests and releases in year 2015

improve the security for browsers, as it was also shown in [19].

Nevertheless, the experiments show that a certain amount of users does not immediately install the latest updates. For example, several requests were logged with a Firefox of version 37, after the release of version 38, as it is shown in Figure 4.4.

The delay in the update process represents a critical security weakness, since the release of Firefox 38 resolved 15 vulnerabilities in the earlier versions, which are also publicly available at this point [44]. These 15 vulnerabilities are documented in the release notes, which can be explored by attackers. Even though the update resolves the vulnerabilities, the attackers can still abuse those, because of the delay in the update mechanism. A similar observation could be done for the releases of Firefox 39, 41, and 42, which resolve 13, 19, and 18 security flaws respectively.

The investigation of request from Google Chrome browsers produced similar results. The major releases for Google Chrome have been mapped to the logged events of the web server of HPI-VDB , which is illustrated in Figure 4.5. The Figure shows that updates have been performed in a timely manner, which could be a result of the automatic update routine. Nevertheless, for several releases a

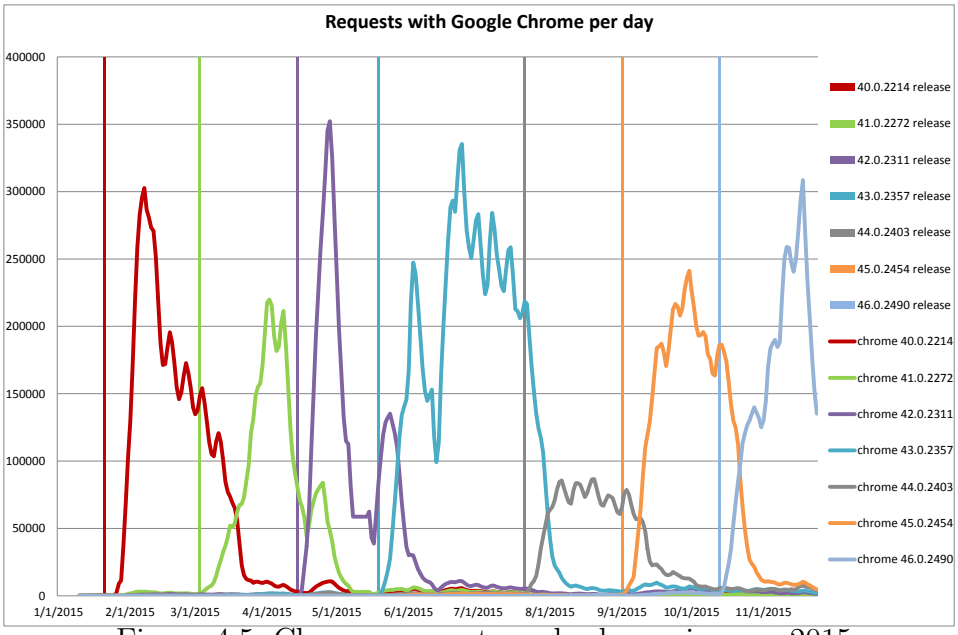


Figure 4.5: Chrome requests and releases in year 2015

time period of approximately one week could be identified, which elapses until the majority of users adopted the new update.

Another possibility to find useful information about running applications that communicate via the network, is the utilization of logs from proxy servers. This possibility omits the necessity to perform a comprehensive scan of the target network and each client for running applications. It benefits from the fact that those applications will exchange some messages via the network. The proxies are points in a network, which are often used as strategic locations for SIEM systems or firewalls. Since these systems usually serve as entry points to a local network infrastructure, the outgoing and incoming network packets have to pass the proxies. Thus, the messages of those applications have to pass the proxy system as well and leave traces in the proxy logs. These traces could in turn be used in a similar way to gather information about active applications. The advantage to the log forwarding is similar to the web server approach, since it is not necessary to forward the logs in the beginning. The information about the clients is already centralized at the web server or the proxy system. One additional advantage of the proxy-based approach is that this approach is not limited to the identification of the browser of the clients. It is rather able to

process information of any application that communicates via the proxy, such as an FTP client or a messenger application. Thus, the proxy-based approach covers a broader range of applications that communicate over the network, although the browser might still be the most used application there. The disadvantage is the lack of completeness as only applications that exchange messages via the proxy will appear in the logs. Thus, if an application is only communicating with internal systems and produces only local network traffic, it will not appear in the proxy logs. Since proxy systems are usually located at network boundaries only the external incoming and outgoing traffic will be recorded there, which could miss several applications. An example of a proxy event that was then recorded by Arcsight, which is a popular SIEM system, can be found in Listing 4.5. The event includes the important items of the proxy log, such as destination and source as well as the application. In the example from Listing 4.5, the application is *http*, which means that the proxy connection contains HTTP requests of a browser. Thus, the requestClientApplication field contains the important part that provides additional details about the application. This part is underlined in the Listing. So, again the user agent can be used to determine the version of the browser. Finally, the browser name and the detailed version string will be used to perform the lookup in the CPE dictionary and derive the corresponding CPE-ID for this application.

Listing 4.5: Sample Log Message of Arcsight

```
app=http src=XXX.XX.XXX.XXX dhost=www.test.com
dst=XX.XXX.XX.XXX request=http://www.test.com:80/someService
requestMethod=POST requestContext=http://www.test.com/index
.html requestClientApplication=Mozilla/5.0 (Windows NT 6.1;
WOW64) AppleWebKit/537.36(KHTML, like Gecko)
Chrome/43.0.2311.60 Safari/537.36 requestUrlHost=www.test.com
```

The application identification on proxy logs was also deployed in a research cooperation with one of our project partners. They run a large infrastructure with numerous servers and clients. It was possible to test and execute the application detection on real-world data that was received through the project contract. However, it was challenging to process the data in a reasonable time, since the

data accumulates to around 100 GB of compressed events and 1.5 billion events per week. Although, the company network is well managed, it was possible to identify some strange occurrences in the log files. The proxy logs were first grouped by the MAC address of the client to allow a backtracking to the individual users later. However, it turned out that several clients use various user agents, which could indicate that the identified MAC address does not belong to a real client, but rather to another router or a switch, which cascades another local network. However, a correlation with the lookup file for all registered clients showed that the hostname of the device indicates that it belongs to a laptop of the company. Besides, this observation the company usually does not allow users to perform manual installations on their devices and they manage the update behavior of each device. Hence, it is peculiar that several clients with outdated user agents have been detected in the corporate network, since they should receive the browser updates automatically.

Therefore, these two approaches show the feasibility to identify communicating applications in a corporate network, without performing extensive scans of the connected clients. Since the information is available nonetheless, because internal web servers or proxy systems are likely to be in place, it can also be used to identify applications and later correlate the corresponding vulnerabilities.

4.5 API for Threat Intelligence Platforms

Another use case for vulnerability information is the interoperability with other services to serve as a knowledge base. This information integration can be utilized in a threat intelligence platform, which is able to correlate information from the vulnerabilities with current or historic events in an infrastructure. Furthermore, the vulnerability information can be used as indicators in threat intelligence platforms to provide the ability to evaluate the security of a target system. If the browser of a specific client contains vulnerabilities, the requests of this browser might have an impact on the overall security of the system. The browser might be exploited directly, without the need of downloading and executing a malicious application but by executing malicious script or by erroneous interpretation of

the website. The logical preconditions of the vulnerabilities allow an evaluation for partial and overall fulfillment. Thus, the events can directly be mapped to those preconditions. The vulnerability information can also be used to highlight insecure systems and monitor those systems closely to detect suspicious activity. This interoperability is achieved via an application programming interface(API) that can be used to query the database for information about vulnerabilities. The API is only accessible to registered users that have to use the API key to perform requests. With the API key, a limited number of automatic requests is possible per day. This limitation should prevent an extensive querying that could overload the web server and result in a possibility to perform denial of service attacks(DOS). Furthermore, several filter mechanisms allow a targeted information retrieval for specific scenarios. The vulnerabilities can be filtered based on the CVSS vector, the CVSS attributes, the last modified date, the identifier, and the CPE-IDs. The most important filtering mechanism for many analytic approaches is the CPE-ID filtering, since the approaches usually detect the applications first. Then, the identified applications have to be checked for correlated vulnerabilities. Thus, the API has to provide a filter for multiple CPE-IDs. Each vulnerability that is correlated to at least one CPE-ID is returned in the results, but duplicates are eliminated before returning the results. The results themselves can be returned in different formats that can be chosen when querying the API. The predefined formats are either XML or JSON, which are both well-structured and allow a straightforward integration into other applications. The number of results per request are also limited, which produces a paging of the overall results. The link for the query to the following slice of vulnerabilities will be generated and appended in the *meta* section of the results. A possible extension of this approach would be the integration of the Structured Threat Information eXpression(STIX) [2]. STIX is a widely adapted format to specify any type of threat information and be able to exchange the information across different organizations. Two external companies that query the HPI-VDB continuously are currently using the API for information about vulnerabilities. Additionally, the API is also used for the Attack Graph Construction Platform(AttGCP) [9], which was explained in Section 4.2. Thus, the API is an integral part of the HPI-VDB

that is utilized in diverse research projects and industrial co-operations.

4.6 Conclusion

This chapter focuses on several use cases for vulnerability information. Furthermore, it explains the necessity for the vulnerability database that was created and explained earlier. Especially, the newly created preconditions and postconditions that are available in a machine-readable format are fundamental for different analytic approaches. In the first use case the vulnerability information are used to generate attack graphs from a scenario description. Therefore, two different methods to create the attack graph have been implemented. One method utilizes a graphical representation of the scenario description and highlight vulnerable systems and possibly tainted connections from those systems. Thus, the attack graph itself has a low complexity and directly maps the underlying network structure to the graph. The other attack graph creation utilizes the MulVAL reasoning and creates a complex graph that visualizes possible attack paths in the network. This graph creation uses the vulnerability database as a knowledge base to identify the existence, the requirements, and the impact of the vulnerabilities. The graph creation is performed on an external system that queries the database via API. The following use case represents the most obvious usage of vulnerability information, which is the detection of vulnerabilities on a target infrastructure. Hereafter, different vulnerability detection mechanisms are explained that target separate user groups. To begin with, the average user is taken into account that should also be able to benefit from the vulnerability database. Then, the comprehensive detection of vulnerabilities for single systems is explained. However, some limitations of this approach, such as privacy concerns and the sensitive type of information, are discussed. Thus, a moderate version of this approach was required to provide the service to average users. The browser-based approach was an appropriate solution for this challenge, since it utilizes information, which are already disclosed by the browser of the user. Furthermore, the browser is considered the most crucial application, since it is normally used to connect to remote resources. Attackers can abuse those connections as the initial point of

attack. For the second group of target users, corporate architectures are considered that already collect several information or contain internal servers. Those circumstances allow a passive vulnerability detection approach, which benefits from already generated or collected information. The first part of the passive vulnerability detection detects CPE-IDs based on log information from the individual systems. Since this information is usually generated anyway, it can be used to identify running applications for each target system. Additionally, log information are used for the second type of the passive vulnerability detection as well. The difference is that the information are extracted from web servers or proxy systems. The scalability of these approaches can be enhanced if the log information is forwarded to a central point of the network, which can then perform the mapping to CPE-IDs. The final use case for vulnerability information is based on the API of the database. The application programming interface allows a straightforward integration of the desired information about vulnerabilities, which can be queried from the API. The filtering possibilities of the API can be used to search for information about specific vulnerabilities, which are then returned in a structured format. The possibilities to create different endpoints that provide different representations of the vulnerability information allows a tailored vulnerability model for each use case. These models can be integrated into threat intelligence platforms or in other third party software that requires information about vulnerabilities.

Chapter 5

Vulnerability Data Quality

5.1 Introduction

One major challenge in the development of the HPI-VDB was to ensure the data quality. The problem is that the data is available in the Internet and can be found in different formats and sources. However, especially textual reports about a vulnerability are hard to evaluate in a large scale. Nevertheless, further analytic approaches require a high quality and accuracy of data, since the approaches can only achieve correct results if the underlying data is correct. Thus, for many sources the lack of a verification, which results in low trustworthiness of the information is a problem. Therefore, the NVD [46] is considered as the main source that is queried for vulnerability information. The fact that NVD first receives the vulnerability reports and then completely evaluates the information before a full publication of the vulnerability ensures that the information was verified and approved by experts. So, for the HPI-VDB reliability scores are assigned to each source that should be integrated, as it is illustrated in the model in Figure 3.1.

Another problem is missing or diverse information for the vulnerabilities. If a new vulnerability was found in one of the sources, but the source only provides a textual description of the vulnerability or a subset of the different attributes, the remaining properties have to be integrated later. As it was explained in Section 3.2 the minimum requirements for a new vulnerability are the existence of any kind of identifier and at least a textual description. During the research,

several methods that can deal with missing information have been created. The different methods range from simple correlation to other sources to more complex procedures that automatically deduce properties from other attributes of the vulnerability. All these methods had the major requirement to work in an automated way, as a manual investigation or an adjustment by experts is not feasible for this high amount of vulnerability information.

5.2 Information Enrichment and Correction

This section describes the solutions to missing and defective properties in the vulnerability information. While missing properties can be easily detected and operations to restore the information can be initiated, the issue is different with defective information. For a successful correction of the information, it is necessary to identify the erroneous property first. This requires either a possibility to deduce the property from other attributes or the possibility to compare the wrong information with another source that is known to be more reliable. In addition to defective information with regards to content, it is also possible to find and correct information that was provided in the wrong format. Especially, in the case of vulnerabilities many standards are used to identify applications, weaknesses or other attributes. This standardization allows a verification of the structure of the information and usually it is possible to correct the flawed attribute. Even if the standardized format was not used in the first place, it should be integrated since the analytic approaches often rely on those standards.

5.2.1 Integration of external Identifiers

A typical example of information enrichment are external identifiers. Many providers of vulnerability information do not list additional external identifiers or only include a subset of identifiers. Certainly, it is not possible to always include all identifiers, since there is no finite number of those identifiers. However, several standards and well-known information sources have established themselves in the community over time. Especially those standards are the focus for the

integration of external identifiers. In addition to the lack of those identifiers, the information is often included as plain text information without further explanation or guides how to make use of this identifier. Therefore, expert users are the only ones that can make use of this information, since it requires additional knowledge to achieve anything from those identifiers. Thus, it is a better idea to also include links to the resource or even incorporate the interesting and most important information from the external source. Then, average users can also benefit from the information and the identifier provides a benefit in general.

A prominent candidate that is often missing from the vulnerability information in many sources is the exploit-id, which is the identifier from the exploit-db [49]. This ID usually refers to an exploit of the corresponding vulnerability, which is often mainly interesting for attackers. However, the information from the description in exploit-db can include additional information about the vulnerability itself and possible reasons and circumstances that are required for this vulnerability. Furthermore, the exploit usually contains information about possible targets, which means specific version numbers or operating systems that are required for the exploit and consequently for the vulnerability.

The *common weakness enumeration*(CWE), which was already explained in Section 2.3.4 allows the classification of the vulnerability. The CWE-ID can also be used to identify additional requirements to exploit the vulnerability and it can reveal possible security methods to fix the vulnerability. So a possible cross site scripting vulnerability that is related to the CWE-ID *CWE-80* can be enriched with information from the CWE entry. The corresponding CWE entry contains information about different types of mitigation techniques that can be used by the service provider to resolve the vulnerability.

The *common attack pattern enumeration and classification* is a standardized identifier for attack pattern, as it was explained in Section 2.3.6. It provides some information, which is similar to the CWE-ID, such as mitigation techniques. However, the CAPEC-ID provides additional information, such as severity or possible results of a successful exploitation, which could be cookie stealing or content spoofing for CAPEC-ID 63, which is the CAPEC identifier for *cross site scripting*.

The addition of identifiers usually improves the comprehensiveness of the vulnerability information as it nearly always adds additional information. Thus, the external identifiers should be included in a way that those third party sources can be used to enrich the vulnerability with additional information and provide benefit to the users. Therefore, whenever it is possible the resource should be directly linked and important information can be directly appended to the vulnerability to allow a full utilization of that information for analytic approaches.

5.2.2 Solution Creation

As it was described before, the analysis of vulnerability information relies on information about the affected application. Furthermore, additional requirements have to be fulfilled to ensure the existence of the vulnerability. However, one question remains. When the vulnerability was successfully detected and the existence validated, the responsible administrator has to resolve the vulnerability. Since it does not provide so much benefit to only detect the vulnerabilities and compose a report about the existence of numerous vulnerabilities on a system without any further information about possible solutions, one important piece of information is the solution for a vulnerability. The vulnerability model that was presented in Section 3.2 already includes a solution property for each vulnerability. Therefore, this information has to be extracted if the specific source for the vulnerability information incorporates insights about possible solutions. However, several sources do not propose information about solutions for vulnerabilities, because it is often difficult to investigate the causes of a vulnerability and identify possible workarounds. Another limitation is the high amount of vulnerability reports that do not allow a manual and time-consuming investigation for each vulnerability. These limitations were the reason for the development of the automatic solution creation, which was also presented in [23]. The feature was implemented in the HPI-VDB [27] and is applied for every import of vulnerability information. The solution creation first checks if the source has an attribute that contains information about possible solutions. If it is not the case, the algorithm tries to derive the necessary information from the vulnerability

description. Since the vulnerability description is a mandatory attribute of a vulnerability, as described in Section 3.2, it has to be available for this use-case. In addition, also humans would try to extract the missing information from the description of a vulnerability. Therefore, it was an obvious step to utilize the description for an automated approach as well. The challenge for using the vulnerability description is the natural language of the description. It is designed to be understandable for a human expert and therefore not directly usable for an automated approach. However, several vulnerability descriptions have been analyzed in the preparation of the solution creation approach. The investigation revealed certain patterns that occurred regularly, when a vulnerability of a specific software version was detected and a newer version already fixed the problem. Usually when a vulnerability in an application is detected, it affects either all older versions of the application, which have been published before the patch or it affects the versions that incorporate the newly integrated feature. One of the identified patterns originates from the fact that the keywords *before* or *after* appear directly after the name of the program. Thus those patterns could be used to construct regular expressions that can be applied on the vulnerability descriptions and yield the software version with the corresponding fix. The regular expressions were divided into two different parts that occur usually one after another, as it is illustrated in Listing 5.1. The first part is used to identify the name of the application, which should also appear in the preconditions. Since the existence of the application is a necessary requirement for the vulnerability, this fundamental precondition includes the name of the application as a part of the CPE-ID, as it was explained in Section 3.2. The affected program name is used in connection with the new version that resolves the vulnerability. Then this application name can be used to verify the match of the regular expression, as it has to appear in the preconditions. Therefore, the match, which will be returned in the first group of the regular expression for the application name, has to be identical with the application name in the corresponding CPE-ID. Furthermore, the application should appear in an enumeration, which could be identified with the keyword *and* or a *comma*. It could also appear at the beginning of a sentence or directly after the keyword *in*. So the final regular expression to detect the

application is constructed according to these requirements and is illustrated in Listing 5.1. In this case, the description provides additional information about affected versions and about a version that does not include the vulnerability anymore. Then this version identifier occurs behind the regular expression for the application name and the keyword, which is either *before* or *prior to*.

Listing 5.1: Regular Expressions for Solution Creation

```
Application Name: (?:^| in |, and | and |, )(.*?) (?:before|
    prior to)
Version: ((?:Build |Patch )?(?:SP|\d)[\d\.a-z\(\)\-_]+)
```

The investigation of numerous vulnerability descriptions in the HPI-VDB revealed that the version identifier could be composed of various characters. Therefore, the regular expression has to be a bit more specific, since the following word in the description is not known. The difference to the expression for the application name is that this expression benefits from the fact that the following term is known and can be used as a boundary for the application name. Thus, the regular expression for the application name can be composed with a minimum number of wildcard characters, followed by the keyword *before* or *prior to*. It is not possible to use the same principle for the regular expression of the version. Therefore, the specification for the allowed characters in the version itself has to be detailed and tailored. The result of the investigation showed that the version can include a *Build* or *patch* keyword followed by the specific number for the build. Another possibility is the specification of a service pack identifier, which is often used in the context of operating systems. The most common possibility is a combination of numbers, lowercase letters, dots, dashes, underscores and brackets. These insights result in the regular expression that is illustrated in Listing 5.1.

After this method was designed and tested, it was directly implemented into the HPI-VDB. It is applied on all incoming vulnerability descriptions and creates the solution objects as it was described in Section 3.2. The HPI-VDB contains information about 99,381 vulnerabilities and a total number of 52,594 solutions.

However, the solutions are only available for 31,420 vulnerabilities[1], because some vulnerabilities can have multiple ways to be resolved. Nevertheless, 31.6% of the vulnerabilities contain information about possible solutions and all those solutions have been created with the described method in an automated way.

Another possibility to include information about solutions for vulnerabilities is to evaluate the references of a vulnerability, if the information is available. This was shortly introduced in Section 3.2 and requires additional workload during import time. The specified references will be crawled and if the website contains information about solutions, the corresponding textual information will be embedded into a solution object and stored along with the vulnerability. However, this approach is limited due to the various formats of third party references. The crawler can parse the website and check for the *solution* keyword in a prominent place, such as headlines, beginning of paragraphs or on a single lane followed by colons. In these cases, the information can be retrieved, but often it is not directly possible. Another problem is the legal requirements. Many providers do not allow an automated retrieval of their sources or only permit the information processing for specific clients or after a user logon. These restrictions affect this possibility as well and therefore it is easier and more convenient to derive the information from textual description. This derivation does not affect the import performance as much as the crawling and evaluation of external references. It is also similar to a human that tries to extract the necessary information from the description. Since the vulnerabilities are still reported by human investigators most of the information can be found in the description.

5.2.3 Correction of CPE Identifiers

Another part of the information correction focuses on the correction of CPE identifiers. As it was already explained in Section 2.3.2, the CPE identifiers are used to identify applications with a high level of detail. The MITRE Corporation provides the information itself and it is also responsible for the publication and the

[1]Statistics from 16th of April 2018

maintenance of the CPE dictionary. The CPE dictionary contains all currently available entries for the analyzed applications. However, the schema description allows creating CPE identifiers for unlisted applications on one's own by following the specified guidelines. The necessary information to create the CPE identifier consists of the application name, the application type, which is operating system, hardware, or normal application, the vendor name, and the version identifier. Additional information can be appended to the identifier to specify further details, such as language packs, architecture, patch number. The main problem is that not all providers of vulnerability information completely follow these instructions, which could lead to inconsistencies. Another possible source of those inconsistencies are changes over time, when the CPE dictionary changes and specific entries are deprecated. Then the information providers should also reconsider the adjustments and integrate the changes, which unfortunately does not happen often. In the end, several information providers still work with outdated CPE identifiers, which could be problematic for analytic approaches. The difficulty is that CPE identifiers are interpreted as strings that are associated to a specific application. If two CPE identifiers differ, then the two associated applications should also be two separate applications. Thus, a two-tiered error correction feature was integrated into the HPI-VDB. In the first part, the CPE dictionary is integrated and will be used to find and correct the deprecated CPE entries. In the second step, the remaining CPE objects are checked for tuples of similar entries, which are likely to relate to the same application.

In the first part, an iteration through the complete dictionary is performed and the deprecated entries are extracted and put into a dictionary structure for a fast access later. Then those deprecated entries are checked against the database. If no match can be found the respective CPE identifier can be discarded. Otherwise, the found deprecated CPE will be tested for the existence of the corresponding updated CPE entry. If it can be found, the pair will be appended to a list of results. In the other case, the updated entry has to be created. This additional test is necessary, since the updated CPE identifier could already be listed in the database, if it was introduced by another more recent vulnerability, and the strings for the CPE identifiers should be unique. Finally, the list with the

identified tuples will be returned for logging and traceability purposes.

Algorithm 2 Identification of deprecated CPE identifiers

```
deprecatedCpe = { }
foundMatches = [ ]
for each entry in CPE_DICT do
    if entry is deprecated then
        deprecatedCpe[entry] = entry.deprecated_by
    end if
end for
for each entry in deprecatedCpe.keys() do
    try retrieve CPE object with entry.name
    except CPE does not exist
        continue

    try retrieve CPE object with deprecatedCpe[entry]
    except CPE does not exist
        create CPE object with entry.name

    foundMatches.append([deprecatedCpe[entry],entry])
end for
```

The second part iterates through all CPEs in the database and searches for other CPEs that have a similar vendor name and a similar application name. This means that the vendor and the application names should be identical if they are transformed to a lowercase version. If the algorithm could find some candidates that fulfill these conditions, the two version strings are checked for inclusion. Therefore, the CPE dictionary is used as the gold standard to obtain the CPE, which is officially documented in the dictionary. Then the two similar CPEs are merged and the correct entry from the dictionary is kept, whereas the other CPE is deleted from the database. However, prior to the deletion of the wrong entry, all relations have to be adjusted to preserve the relationship between application and vulnerability.

Some example results that have been found with this approach are illustrated in Table 5.1. Usually the version string is mixed up and the attributes are stored in the wrong position or the character to indicate a blank spot in the

CPE identifier was changed from a "-" to no character. Another frequent source for those similar CPEs is the inaccuracy with the type of CPE, which is often between **a**pplication and **o**perating system. Although most often the amount of vulnerabilities that are associated to the detected CPEs is not that high, it is still important to be able to match those vulnerabilities with the correct CPE. Otherwise, those vulnerabilities will never be considered, since the analytic algorithm will only detect the official, not deprecated CPEs and their corresponding vulnerabilities.

Table 5.1: Sample of similar CPEs detected

CPE1	vulnerabilities of CPE1	CPE2	vulnerabilities of CPE2
cpe:/o:microsoft:windows _2003_server:enterprise	2	cpe:/o:microsoft:windows _2003_server:::enterprise	1
cpe:/o:microsoft:windows _xp:-:sp1:home	2	cpe:/o:microsoft:windows _xp::sp1:home	107
cpe:/o:cisco:ios:-	13	cpe:/o:cisco:ios	14
cpe:/a:opera:opera:5..10	7	cpe:/a:opera:opera:5.10	4
cpe:/o:openbsd :openbsd:3.1	25	cpe:/a:openbsd :openbsd:3.1	1
cpe:/o:google:android:1.1	4	cpe:/a:google:android:1.1	1

5.3 Validation of CVSS Attributes

The completion of the information for the vulnerability model includes the addition of CVSS attributes as well. The *Common Vulnerability Scoring System* information is valuable for many analytic approaches, as the information about violations of the different security goals can give insights about the type of the vulnerability. Nevertheless, the attributes are missing in several sources for vulnerability information or the information is appended at a later point. This delay occurs, because of additional tests and checks that have to be performed by vulnerability analysts, which also start with the textual description of the vulnerability [18]. Then the experts have to perform this manual investigation of the vulnerability to assign the CVSS properties. The time delay could be

crucial for analytic approaches that rely on those attributes. The delay of the scoring and evaluation of vulnerabilities could amount to several days, as it was the case of the OpenSSH vulnerability with the identifier CVE-2016-0777. It was published on 14th of January 2016, whereas the CVSS attributes have been added on 19th of January. This delay of five days limits the capabilities of many analytic approaches to process the information for this vulnerability. However, attackers can already benefit from the publication of the vulnerability, because they usually rely on the textual description. Thus, this advantage of the attackers can be limited with an automated or semi-automated attribution method. Then, the manual investigation of vulnerabilities is not necessary in general or at least for some cases. This approach should consequently reduce the workload for the vulnerability analysts and create the CVSS attributes directly with the initial publication.

The automated vulnerability analysis is based on a classification of the vulnerability according to the textual description. So, the necessary information to perform the classification is always given, as the textual description is a mandatory part in all vulnerability databases or submission forms [7]. The idea of an automated classification of vulnerabilities based on their textual description was also presented in [58] where the authors want to predict vulnerabilities in software components before the release. However, they face the problem that the normalization of the different vulnerability sources requires a large effort. A similar approach was presented in [5], which focused on the exploitability of different vulnerabilities and used data mining approaches in vulnerability databases.

The classification methods that are necessary to predict and derive the CVSS attributes have a more promising perspective, since the classification results should yield whether a security goal was violated or which attack vector was detected. It is more likely to be able to derive those results from the textual description than the exploitability of future weaknesses. The classification method focuses on the prediction of the most important CVSS characteristics, such as violation of the security goals and the attack vector. Thus, the classifier has to determine the value for the attack vector, which could be *remote*, *local*, *adjacent network*, or *unknown*. Furthermore, each of the security goals, namely availabil-

ity, confidentiality, and integrity, has to be evaluated and it has to be determined whether and how those goals are violated. Those values could be *None*, *Partial*, or *Complete*. Finally, the classifiers should be able to predict those four attributes either independently or as a combination. The combination results in a maximum of 108 different categories that can be used. The classification of the vulnerabilities was performed with two different methods, namely *Naive Bayes* and *Neural Networks*, which will be explained in the following. The classifiers have been evaluated on the available vulnerabilities in the HPI-VDB and the results will be presented in Section 5.3.3.

5.3.1 Naive Bayes Approach

The first approach that was used for the automatic classification of vulnerabilities is based on a naive Bayes classifier [54]. The choice for the naive Bayes classifier has been made, because the naive Bayes approach is a widely known and commonly used algorithm for classification problems. Thus, the approach has to classify the vulnerabilities based on the corresponding description according to the selected CVSS attributes, which should be determined.

However, some preparation steps are required to apply the classifier on the vulnerability descriptions. The first preprocessing step is to determine the most meaningful words that can have the highest impact on the classification. This determination depends on the selected feature and this step has to be performed for each of the possible attributes. The processing applies a bag-of-words model that represents the existence of individual words. The bag-of-words model is also widely used in the domain of document classification and was a natural choice. Nevertheless, the model requires the specification of an overall dictionary of all possible words. The dictionary creation should already consider the classification domain, since different domains have a different vocabulary. In this case, the dictionary was created with the available vulnerabilities in the HPI-VDB. Therefore, around 72,000 vulnerability descriptions have been exported from the HPI-VDB. Those descriptions were analyzed and resulted in a list of around 104,000 words. Thereafter, the descriptions are normalized, which means the descriptions are

cleansed from stop-words [64] that do not provide additional information. Another step in the process of normalization is the stemming of the single words of each description. The stemming removes differences that only occur, because of declination and conjugation. Those differences do not provide additional information as well, but will result in multiple words for the same information. Up to this point, the preprocessing steps are identical regardless of the selected attribute. However, the next step will depend on the selected attribute that should be considered for the classification. This step is a first training round that is used to find the most meaningful words. Those meaningful words are directly bound to the selected attribute, since the violation of the integrity would have different important keywords than the attack vector. To find the most important words for the selected attributes, all words are ranked by their importance, which is measured by the impact of the existence of the word to the classification result. Thus, the prepared and cleansed descriptions are separated based on the value of the selected attribute. Therefore, the descriptions will be divided into specific subsets for each of the values of the attribute. If the attack vector is considered for example, the descriptions are separated into 3 subsets of vulnerability descriptions with a *remote*, a *local*, and an *adjacent network* range. Then those subsets are used to compute an adapted form of the relative term frequency and an adapted version of the inverse document frequency for each of the words. The relative term frequency is computed as the fraction of the number of occurrences of a given word or term (t) in the document (d) and the maximum number of occurrences of any word (t') in the document. Then the fractions are summed up over all documents in the subset of the value of the selected attribute as it is illustrated in the equation 5.1. The inverse document frequency is computed as the logarithm of the fraction of the number of all documents without the subset of the value of the selected attribute ($\overline{subset_{attributeValue}}$) and the number of documents in this set that contain the given term (t). The equation for the second statistic can be found in 5.2. In the specific case of vulnerability descriptions, each description is considered as a single document.

$$TermFrequency(t,d) = \sum_{d \in subset_{attributeValue}} \frac{|\{t \in d\}|}{\max_{t' \in d} |\{t' \in d\}|} \quad (5.1)$$

$$InverseDocumentFrequency = log \frac{|\overline{subset_{attributeValue}}|}{|\{d \in \overline{subset_{attributeValue}} : t \in d\}|} \quad (5.2)$$

The relative term frequency is a measure to find candidates for important words, since it is more likely that a word has a higher impact on the value of the selected attribute if it appears more often in the corresponding subset. On the other hand, it is also more likely that a word is specific to a value of the selected attribute if it does not appear so often in the complementary set. This means that a word that appears only in one of the subsets that was created by the separation based on the value of the selected attribute has a strong impact on this classification. This observation can also be explained with the already mentioned effect of domain specific vocabulary and keyphrase extraction [17]. In natural languages, one usually uses a specific vocabulary to describe scenarios in a certain domain. Thus, it is possible to evaluate the utilized vocabulary to derive the domain and try to predict the meaning of the description. Therefore, a group-wise comparison of the term frequencies and a selection of the words with the highest discrepancies was performed. The resulting candidates have been investigated manually and they looked promising, as each of the candidates could be explained and a connection to the value of the selected attribute was possible. For example, words, such as “remote”, “message”, or “connection” have a strong indication that the attack vector of a vulnerability is *remote*. Whereas, words, such as “crash” or “denial” suggest a violation of *availability*. So the candidates for the most promising words have been selected based on their term frequency and their inverse document frequency. Therefore, the inverse document frequency and the term frequency should have high values for words that have a strong influence on the value for the selected attribute. Then the possible candidates have been sorted according to their term frequency and the inverse document frequency, as shown in the equations 5.1 and 5.2 respectively.

For the final classification round it proved to be sufficient to use the 500 most meaningful words to achieve satisfying results. The limitation to 500 words arose from the performance limitation, as the approach should be able to be executed on a web server without significant performance losses. Therefore, this initial training round considered all of the 104,000 words and revealed the 500 words that provided the highest impact to the classification.

The resulting list of words has been investigated manually as well, since 500 words is an amount that is still manageable for a human. The proposed candidates of the training round are comprehensible. Then the final features that are used by the naive Bayes approach are binary representations of the existence of those 500 words. Therefore, an iteration over all vulnerability descriptions is performed that flag the existence of one of the 500 words. This execution was only possible, because of the limitation to 500 words, as a binary vector over all 104,000 words would require memory of roughly 13 megabyte for each vulnerability description, which is not feasible if the classifier has to deal with 72,000 vulnerability descriptions. The prepared features that amount to a 500-bit list for each vulnerability description are then passed to the classifier for the training of the model.

The results of the naive Bayes approach are illustrated and discussed in Section 5.3.3, since they are compared to the Neural Network approach that contains similarities to the naive Bayes approach but also some special characteristics. Thus, the next Section describes the Neural Network approach in more detail.

5.3.2 Neural Network Approach

The second approach that was chosen and implemented utilizes neural networks to classify vulnerability descriptions. The classification problem is identical and the approach starts with the vulnerability descriptions in natural language as well. Neural networks are also widely used for multi-class classification problems that have a high complexity. Therefore, it was another natural step to apply a neural network based classification. The approach should be able to deal with natural language, as recent research revealed good progress in the implementa-

tion of chat-bots that derive meaning from the natural language and propose appropriate answers. A requirement to apply neural networks is that the classifier has to be trained on a supervised dataset in advance, which provides the required labels for the training. However, this limitation is easily overcome by the dataset that was exported from the HPI-VDB. The other condition, which is a known vocabulary that can be interpreted by the classifier, can also be met by using the dataset of the HPI-VDB. The 72,000 provided vulnerability descriptions can be analyzed and transformed into a dictionary of all included words, which can then be used as the vocabulary for the domain of vulnerability descriptions, similar to the naive Bayes approach. The necessary steps to prepare the vulnerability descriptions and transform them into a format that can be fed into the neural network are described in the following.

The first step of the preparation part is the removal of unnecessary information from the vulnerability descriptions. Since the description is composed in natural language by a human investigator, it contains numerous additional words that are only needed to create well-formed sentences but do not contribute additional meaning. Thus, those additional fill-words can be filtered out as they increase the complexity for the neural network without adding information. Therefore, the filtering will not have an impact on the available information. As it was described in the naive Bayes approach the first filter targets well-known stop words [64]. Thereafter, conjunctions, prepositions, auxiliary verbs, and personal pronouns are omitted. In a next step, the remaining words are transformed into their principal part. This removal of differences that are produced by declination and conjugation is also called word stemming. The information of the word itself will not be changed. The last step of the word-by-word processing is the transformation to lowercase letters to remove differences that arise from the position of the word inside the sentence. These three steps remove unnecessary words and transform the vulnerability descriptions into lists of substantial words. The transformed descriptions can still be read and interpreted by humans, although the grammar is incorrect. Nevertheless, those transformations are necessary, since the remaining differences between the words are related to the information content and therefore the classifier should consider those differences. The three

preparation steps are also represented in Figure 5.1 that illustrates the overall workflow of the neural network approach.

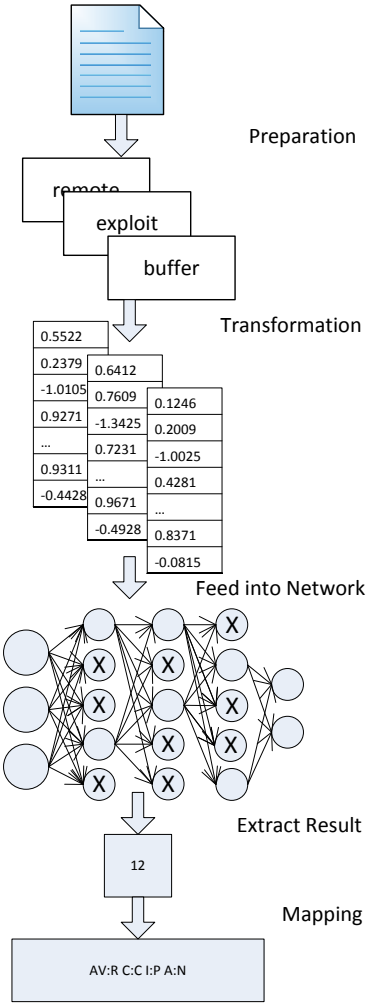


Figure 5.1: Workflow of the Neural Network Approach

The next step of the workflow utilizes the specially prepared vulnerability descriptions and transforms the individual words to a format that is usable for the neural network. This preprocessing step is necessary, because neural networks usually work with numeric value or vectors of floating point numbers. Normally, neural networks are not able to process textual data directly. Thus, a suitable

transformation for the words is needed. The transformation step is illustrated as the second step in the workflow in Figure 5.1 and it is labeled as *transformation*. The specific transformation replaces each of the word with a vector of floating point numbers. For the concrete implementation the GloVe [52] tool was used. GloVe is an unsupervised learning algorithm that derives vector representation for given input words. The distinctive feature of this algorithm is the capability to identify words with similar meanings and to retain the relationship between multiple pairs of words. For example the difference of the vectors for "king" and "queen" and the difference of the vectors for "man" and "woman" have a high similarity. The authors of the GloVe algorithm even state that it is possible to form equations, such as *"king" - "man" + "woman" = "queen"*, because of the relation between the words that was identified with the GloVe algorithm. This means that the example removes the "male" attribute from "king", which would leave something like "royal" and adds a "woman" to receive a "royal" "woman", which is a "queen". The algorithm was applied to the vulnerability descriptions, since it is necessary to retain some relationship between different words as well. Thus, the GloVe algorithm was configured to produce a 50 dimensional vector for each word in a vulnerability description, which was the proposed configuration. The result of the GloVe algorithm is a kind of dictionary that provides a mapping between each of the words in all preprocessed vulnerability descriptions. This dictionary can then be used to replace the words with their vector representations as it is illustrated in Figure 5.1. In the final step each of the prepared vulnerability descriptions, which have been translated to lists of the substantial words, are transformed into lists of the respective vector representations of their words. Thus, each vulnerability descriptions is built of n 50 dimensional vectors, where n denotes the number of words in the description. The final problem that has to be solved is the different lengths of all vulnerability description, since the textual descriptions vary greatly in their length the lists of vectors differ in their number of vectors as well. Therefore, a normalization method has to be applied to adjust the length of the vector lists to a uniform size. In the first proposal, the average number of words was used and each description has to be modified to match the size of 37 words. However, this method had the effect that long descriptions

have to be shortened, which would mean some words had to be omitted. Since this omission has to be performed in an automated way, it is likely that some words that contribute meaning are left out. This option is not possible, as it would have a negative impact on the overall performance of the approach, as the neural network should be able to make use of all available information in the descriptions.

Therefore, a better idea is to use the maximum number of words in the descriptions. Then no description has to be shortened and the risk of an information loss could be circumvented. This means that all shorter descriptions have to be enlarged without modification of the information in the descriptions. The solution is dummy word vectors that are not mapped to a specific word in the vocabulary, but they merely act as placeholders. The used dummy word vector is the null vector, which only includes zeros in all 50 dimensions. The largest vulnerability description consists of around 400 substantial words that are left after the filtering in the preparation step. Thus, each vulnerability description is enlarged with null vectors until the corresponding list of vectors has 400 items. This means that the neural network has to process 400 vectors with 50 dimensions. So, each vulnerability description is transformed into 2,000 input values that are fed into the neural network. At this point, the neural network can process the transformed descriptions, but the classification task should also consider the labels that identify the different classes. Therefore, the labels have to be extracted and converted to a numeric space. For the single attribute classification, a simple enumeration of the different attribute values is sufficient. Therefore, each attribute was investigated and the distinct values have been identified, which are *None*, *partial*, and *complete* for the security goals and *unknown*, *local*, *adjacent network*, and *remote* for the attack vector. In the case of multiple attributes, all possible combinations of different values are generated and enumerated to derive the numeric values. This means that the CIA classification, which determines the values for *confidentiality*, *integrity*, and *availability*, has to deal with 27 different labels, since each of the attributes has three possible values. The other combination additionally considers the attack vector, which results in 108 possible combinations and 108 different labels. Finally, this mapping has to be

persisted, since it should be used to translate the determined label to the textual form, as it is also illustrated in Figure 5.1.

At this point all necessary preprocessing steps have been performed and the transformed vulnerability descriptions with the corresponding label can be fed into the neural network. To train the classifier the prepared dataset is randomly split into a training dataset, which contains 80% of the data, and a testing dataset with 20% of the data. The neural network was configured with a learning rate of 0.001 and a run-time of 100 epochs. Several tests showed that the accuracy of the network converges after this run-time. Furthermore, different configurations for the number of layers and the corresponding units per layer have been tested. The most promising results have been achieved with a network of 3 layers and 1200, 700, and 50 units for the respective layers. The investigation of the results, which will be described in the following, revealed that the algorithm encountered a problem that is known as overfitting. The results in the training set have been accurate with an overall accuracy of around 99%. However, the application on the test dataset produced an accuracy of only 70%. This observation is typical for overfitting of a neural network. It means that the algorithm was trained and adjusted on the training dataset. It does not generalize the problem anymore, but learned the distribution and labels from the training set directly. The usual approach to tackle this problem is the specification of a dropout rate. This rate describes a probability that an individual node of the neural network is kept or dropped from propagation. Thus, not all nodes of each layer influence the output anymore. This method is also illustrated in the Neural Network of our workflow in Figure 5.1. It is shown that the nodes marked with an "X" do not propagate their results to the following layers of the network. Thus, the neural network does not depend too much on the training dataset anymore and the accuracy of the test dataset increased.

In the final step, the numerical labels that have been chosen by the classifier are translated to the corresponding textual values, which are readable for humans. Then, the classification process is complete and either a single attribute value or a combination of attribute values is derived from the vulnerability description. Thus, it is possible to add the CVSS attributes or to check for the correctness

if the source does not have a high reliability. This final step also concludes the overall workflow of the neural network approach, as it is illustrated in Figure 5.1.

5.3.3 Results

For the experiments with the previously explained classifiers, real vulnerability descriptions from the HPI-VDB [27] have been exported. This means that the classifiers had to deal with real world vulnerability descriptions, which are published on different vulnerability databases before 1st of January 2016. Therefore, all available vulnerabilities, which have CVSS attributes and are published before 1st of January 2016, have been selected. Then, the CVSS attributes and the vulnerability descriptions were exported to create the labeled dataset. The fixed point in time was used to always refer to the same set of vulnerabilities, since new vulnerabilities are detected and published continuously, but the results of the classifiers should be reproducible and comprehensible. The work and the experiments with the classifiers already began in the later time of 2016 and early time of 2017, which resulted in the choice for 1st of January 2016. Hence, the resulting dataset contains 72,490 vulnerabilities with their corresponding CVSS attributes and their descriptions. The dataset was divided into a training dataset of 80% of the vulnerabilities and a testing dataset with the remaining 20% of the vulnerabilities. The division of the dataset is performed with an initial shuffling to select random vulnerabilities for each set. This randomness ensures that the results of the classifier are not achieved by using convenient data. The performance of the classifier are independent of the selection and differ for each execution of the classifiers. This difference can result in slightly different results for each execution. Therefore, the results have been determined by multiple executions and the average performance of these executions. Each of the classifiers was trained and executed 10 times and the average accuracy was computed.

Additionally, a third dataset, which will be referenced as *validation dataset*, was created with vulnerabilities that are published in 2016. The validation dataset consists of 2,400 vulnerabilities and is used as an additional evaluation for each of the classifiers. In addition to the test dataset, this secondary

dataset allows a more thorough test of the classifiers as the vulnerabilities in the validation dataset were definitely not used in any of the training steps. This is especially important for the neural network approach, since the test dataset is fed into the network as well. It is used to adjust and fine-tune the parameters in each iteration of the network in addition to the training dataset. Therefore, the descriptions of the validation dataset did not have any impact on the training of the neural network. This means that the application of the trained neural network to the validation dataset is similar to the application of the network on newly discovered vulnerabilities, as it would be used in the future. Thus, the accuracy on the validation dataset is the most important evaluation criteria for the two classifiers.

The overall prediction of the CVSS attributes focused on the most important CVSS attributes, which are the violation of security goals and the attack vector. As it was mentioned in the previous chapter, which described the approaches, the resulting classifications are based on the *availability*, the *confidentiality*, the *integrity*, and the *attack vector*. In the experiments, a separate classifier was created for each of these attributes. Furthermore, additional classifiers for the two combinations of the attributes were built. One classifier combines the violation of the three basic security goals and will be referenced as the CIA classifier, since it classifies the vulnerability based on the violation of *confidentiality*, *integrity*, and *availability*. The second and most comprehensive combination includes the *attack vector* in addition to the security goals. Thus, it contains all of the important attributes and will be referenced as the overall classifier.

The export of the vulnerabilities and the subsequent division into sets for each value of the selected attribute revealed another problem that has to be solved. The different sets vary in their respective size, which results in unbalanced sets for training. This difference in the distribution of each attribute value could negatively impact the classification. The problem is that the classifier could achieve an accuracy of 80% already by assigning the most used attribute value to each vulnerability. For example in the case of the attack vector, the division will create one set for all remote vulnerabilities that contains 80% of the overall vulnerabilities. Therefore, if this inequality is not corrected in the training pro-

cess, the classifier can achieve 80% accuracy by classifying each vulnerability as a remote vulnerability. Thus, this imbalance has to be fixed by reducing the size of large datasets, as it is the usual approach for balancing training datasets. Then the pure majority of the attribute values will not influence the algorithm. Therefore, considering the availability attribute, the 72,400 vulnerabilities are divided into 17,700 vulnerabilities with a *complete* violation, 31,900 vulnerabilities with a *partial* violation, and 22,800 vulnerabilities which are unaffected. Then those sets are reduced to 17,700 vulnerabilities each, which accumulates to 53,100 vulnerabilities whereof 80% are used for the training of the classifiers. Then the training sets are equally balanced at the cost of losing some of the vulnerabilities. Hence, the classification should only depend on the identified semantic in the descriptions and not on the pure majority of the attribute values.

After the experiments with the different classifiers were carried out, the performance of each classifier has to be evaluated. Therefore, the single attribute classifiers were simply compared by their accuracy, which can be computed by a fraction of the correctly classified vulnerabilities and the overall number of vulnerabilities. In the case of multiple attributes, the classifier accuracy is computed over all labels to correctly evaluate the effectiveness of our multi-class classification as described in [57]. Thus, the accuracy of the multi-class classification has to consider all selected attributes, as it was also described in Equation 5.4.

$$CorrectlyClassified = \bigcap_{Attr \in SelectedAttributes} \{v \in Vulns | v.Attr = result\} \tag{5.3}$$

$$ClassificationAccuracy = \frac{|CorrectlyClassified|}{|AllDescriptions|} \tag{5.4}$$

Naive Bayes Results

The first results and the initial impression, if it is possible to create an automatic classification approach for vulnerability descriptions was received by the

naive Bayes classification. The initial choice for the naive Bayes approach was done, because naive Bayes is a commonly used method for classification problems that is also capable to process multi-class classification. Furthermore, the wide distribution of naive Bayes results in the availability of libraries in many languages, such as the naive Bayes implementation in the *Natural Language Toolkit (nltk)*[2] for Python. Then, this allows a first impression of the possibility to create an automated classification for vulnerability descriptions, by implementing the classifier and evaluate the initial results. In addition, the straightforward usage and implementation of the naive Bayes classifier allows to reproduce the results and it could be used as a candidate to compare results from more sophisticated approaches, such as the neural network approach. Thus, the naive Bayes classification was implemented and applied, as it was already described in Section 5.3.1.

The approach produced four separate classifiers for the single CVSS attributes, namely *attack vector*, *availability*, *confidentiality*, and *integrity*. The results for the four classifiers on the different datasets are shown in Table 5.2. The results revealed that the accuracy on the different datasets does not vary much, which was expected since the training dataset is created by a random partition for each execution of the classifier. Therefore, the training dataset and the test dataset produced almost identical results in the experiments. A slight difference can be observed in the performance on the validation dataset, which is completely isolated from the two other datasets, as it was created from vulnerabilities that were discovered later. However, the results are promising as all attributes can be predicted with an accuracy of nearly 70%. Especially the attack vector could even be predicted with a high accuracy of around 90%, which could indicate that the description often contains hints about the necessary attack range.

In addition to the presented results from Table 5.2, the naive Bayes implementation provides a confidence value for the classification result. This confidence value can be used as a measure of the reliability of the classification. Thus, a separate experiment was created with a modified form of the classifier that considers the confidence value. The modified version of the classifier now only

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

Table 5.2: Accuracy of the Naive Bayes Approach

CVSS attribute	train data	test data	validation data
Attack Vector	89.9%	90.8%	92.3%
Availability	68.4%	68.0%	70.0%
Confidentiality	73.2%	72.4%	69.1%
Integrity	74.2%	73.6%	68.3%

propagates the final classification result if the confidence of the classification is at least 75%. Otherwise, the classifier will yield an unknown result for the vulnerability description. This change resulted in similar results as the basic naive Bayes classifier. The accuracy of the *attack vector* drops by 2%, whereas the accuracy of the *availability*, *integrity*, and *confidentiality* increases by 3%. However, these gains come with the disadvantage that 56, 419, 428, and 321 vulnerability descriptions are not classified in their respective group. Therefore, the final accuracy did not increase enough to outweigh the omission of those vulnerability classifications. However, this confidence value could be used when the results of different classifiers are combined to achieve a more comprehensive and sophisticated classification result.

Neural Network Results

The second approach that was implemented utilizes neural networks to classify vulnerability descriptions according to the CVSS attributes. The implementation and workflow was already explained in Section 5.3.2. The neural network was created with Tensorflow of version 0.10[3] and Glove 1.2 [52]. The neural network approach was originally designed to solve the classification for the complex combination of attributes. However, the experiments were also carried out for single attributes, since the neural network approach should be compared to the first approach with the naive Bayes classifier as well. Thus, four different neural networks have been created for each of the four CVSS attributes, namely *confidentiality*, *integrity*, *availability*, and *attack vecotr*. The experiments were

[3]https://www.tensorflow.org/

repeated 10 times to reduce the impact of fortunate initialization. Finally, the average of the 10 iterations is computed and presented in this Section. The accuracy itself was computed according to the formula, which was specified in Equation 5.4. In the case of classifying single attributes, the formula can be reduced to the fraction of correctly classified descriptions and the overall number of vulnerability descriptions. Since the neural network has to be trained in a first phase, the dataset was divided into a training dataset and a test dataset. This division was performed in a similar way to the division for the naive Bayes approach, where 80% of the data was used for training and the remaining 20% of the data was used for testing. The major difference to the naive Bayes approach is that the test dataset is also integrated in the training phase of the network. This integration is performed in every tenth iteration of the training phase. Then the test data is fed into the network to create a preliminary result for the accuracy. Thus, the test dataset is already considered for improving the weights in the network during the training phase. This surely has some impact on the accuracy of the classifier on the dataset, as the classifier already learned from the specific dataset. Therefore, the validation dataset is especially useful to evaluate the performance of the neural network. The network itself was configured to run for 100 epochs. Several experiments with a different number of epochs and the visualization of TensorBoard[4] showed that the different networks converge after 100 epochs at the latest. The other important hyperparameters for the network are the number of layers and their respective sizes. The process of finding optimal values for the hidden layers and the number of hidden layers was accompanied by several experiments, measurements, and adjustments. Different configurations of two up to 10 hidden layers with different unit sizes of the layers were tested. Therefore, a script was implemented that simply iterates through different combinations of layers, trains the network and determines the accuracy of the classification. It turned out that an increasing number of layers does not add much more benefit after three hidden layers, but the execution time will still increase with more layers. Thus, the minimum number of layers with a good performance was chosen, which resulted in three hidden layers. Then, the sizes

[4] `https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard`

for the individual layers were defined with a decreasing number of units for each hidden layer, which was also a result of several experiments that were carried out with the before mentioned script. This means that the first layer consists of 1200 units, the second consists of 700 units, and the third and last layer consists of 50 units. The classification results for single attributes are illustrated in Table 5.3.

Table 5.3: Accuracy of the Neural Network Approach

CVSS attribute	train data	test data	validation data
Attack Vector	99%	88.9%	80.3%
Availability	99%	80.7%	70.0%
Confidentiality	99%	81.1%	70.2%
Integrity	99%	81.9%	69.8%

As it is shown in the Table 5.3, the results for the single attributes vary greatly depending on the dataset. The performance on the training data results in an almost completely accurate classification. This observation points at the effect of overfitting, which was explained earlier in Section 5.3.2. However, the results on the test data are acceptable as more than 80% of the vulnerability descriptions are classified correctly. The application on the validation data results in a lower accuracy, which could be explained by the uneven distribution of the data. As it was explained, the training and the test dataset have been balanced to achieve independence from pure majority influences on the classification result. However, the validation dataset was not specially prepared, which results in an uneven distribution of the different attributes. This could result in a lower accuracy since difficulties of the classifier with special cases can appear more often in the real-world dataset. Furthermore, the style and the language in the vulnerability descriptions could also evolve over time. Therefore, a slight variation in the language of the validation data, which only contains newer vulnerabilities than the training and test dataset, is possible. However, an accuracy of 70% for the violation of security goals is still similar to the accuracy of the naive Bayes approach. The major difference is the accuracy of the attack vector, where the neural network can only achieve 80% compared to the 90% of the naive Bayes. Nevertheless, the overall performance of 70% or respectively 80% for the

single attributes is a proof that the classification in general can be automated. Thus, the approaches should be integrated in the analysis process of vulnerability descriptions and they could speed up the investigation procedure.

Results for Combined Attributes

The previously described classification results show that vulnerability analysis can be automatically performed with promising results for single CVSS attributes. The two approaches are capable to classify the exported vulnerability descriptions based on the selected CVSS attribute. The two approaches produce comparable and similar results. Therefore, neither the naive Bayes approach nor the neural network approach achieved significantly better results. However, the original idea was to create a classification approach that is able to derive all of the important CVSS attributes from a vulnerability description. This would allow an immense reduction of the necessary workload that has to be manually supplied.

Therefore, two different combinations have been identified that are mostly used for further analytic processes with one combination being a subset of the other. The smaller combination consists of the violation of the three security objectives, namely *confidentiality*, *integrity*, and *availability*. This combination is often referenced as the *CIA* objectives, which is derived from the initial letters. So, the results for this combination will also be referenced as the CIA classification results. The second attribute combination additionally includes the attack vector to the CIA attributes. Thus, all four CVSS attributes are combined and the classification approach should derive the full classification from the vulnerability description. A naive approach could be a combination of the single classifiers. The accuracy could be predicted by the product of the accuracy of each of the single attribute classifiers. This would result in an accuracy of $0.9 * 0.7 * 0.69 * 0.68 \approx 0.2956$ for the naive Bayes approach and an accuracy of $0.8 * 0.7 * 0.7 * 0.7 \approx 0.274$ for the neural network approach, if the complete attribute combination is considered. However, an accuracy of only 30% or even only 27% is not satisfying. Thus, the alternative solution was to create new classifiers that learn based on the attribute combination. This means that all

possible results of the combination of all attributes, which will be referenced as *overall* classification, and the CIA classification are investigated and considered. This different value combination will then be used as the different labels for the classification approach. Thus, 27 different labels for the CIA classification and 108 different labels for the overall classification are created. Thereafter, the approaches are similar to the single attribute classification, which means the dataset will be balanced and divided into a training dataset and a test dataset. Then the training of the qualifiers was performed and the trained model was applied on the test dataset and on the validation dataset. These steps were performed 10 times and the averages of these 10 iterations are used as the final results, which are also illustrated in Table 5.4.

Table 5.4: Accuracy of Naive Bayes and Neural Network on Combined Attributes

CVSS attribute combination	Naive Bayes	Neural Network
CIA Test Data	63.9%	71.2%
CIA Validation Data	51.6%	53.4%
Overall Test Data	61.4%	59.3%
Overall Validation Data	48.1%	49.1%

The results show that the creation of a new classifier produces better results than the combination of the different single attribute classifiers could achieve. The neural network approach had an accuracy of 27% for the combination of the single attribute classifiers compared to the 49.1% of the newly created classifier for example. Even though the results are mostly near the 50% mark, it is still remarkable that a classification with 108 different labels is correct for half of the vulnerability descriptions. It also shows that the neural network approach slightly outperforms the naive Bayes approach, with exception of the overall test data. This was partly expected since neural networks are usually applied for problems with a rather high complexity and the complexity for this classification problem increases when combinations of CVSS attributes are considered.

The final observation of these experiments is that the classification of vulnerability descriptions can be automated and even all relevant CVSS attributes can be successfully predicted. Thus, it is possible to use machine-learning tech-

niques for an automated vulnerability classification to save processing time for vulnerability experts.

5.4 Scoring Scheme

The scoring of vulnerabilities has always been an important task in the field of vulnerability analysis. The problem is that the total amount of vulnerabilities is hardly manageable anymore. Therefore, a possibility to rank important vulnerabilities is needed. The measurement to decide which vulnerabilities are more important than others is usually the effect a vulnerability could have on a system. If the impact of a successful exploitation of a vulnerability is higher than this vulnerability should receive a higher weight in the analysis and especially it should be resolved with a higher priority. The currently used scheme to weight and rank vulnerabilities is the Common Vulnerability Scoring System (CVSS) standard, which was described in Section 2.3.3. In the current version of the CVSS the base metrics, which are constant over time and independent from the user environment, are divided into exploitability metrics and impact metrics. The impact metrics consists of the already mentioned violation of the three security goals, *integrity*, *confidentiality*, and *availability*. The exploitability metrics include the *attack vector*, the *attack complexity*, the *privileges required*, and the *user interaction*. Beside the *attack vector* that was already discussed and introduced earlier, the other parts of the exploitability metrics focus on the likelihood and the ease of an exploitation. Thus, these base metrics combine the possible impact and the possibility of a successful exploitation and are used to compute the CVSS base score. The overall ranking ideas are also used in the HPI-VDB. In the database, the CVSS Score is employed to sort the vulnerabilities in the results of the analytic approaches, since usually the results contain numerous vulnerabilities that should be ranked according to the corresponding criticality.

In addition to the pure impact and the severity of the vulnerability, the fact whether the affected application is running and listening should be considered as well. Usually vulnerability scanners will only include vulnerabilities for detected

applications, which often relies on the fact that the applications have to be executed at the time of the scan. One of the most famous vulnerability scanners is Nessus [63], which is often executed from a central point in the network and thus scans the attached clients for vulnerabilities. Therefore, Nessus, as well as most other vulnerability scanners, starts with a port scan to find open ports and derive the listening applications. Nessus can use integrated port scanners or external tools, such as nmap [35] or amap[5] for this task. Then the scanning mechanism is able to determine the corresponding vulnerabilities for each of the identified applications and the listening port. Furthermore, some scanners even try various exploits, if the exploits are publicly known, to test if a successful exploitation of the vulnerability is possible. The benefit is that the scanning mechanism works in a fast manner and it is easy to integrate additional clients that are connected to the network. The only necessary adjustment is that the scanner has to be configured to also include the new client into the detection process. On the other hand, the disadvantage of network-based vulnerability scanning is the inaccuracy, which comes with the limited insights that the scanner can obtain. The scanning tool can only investigate the target from a remote point of view, which is limited to open ports and possible responses for requests on these ports. Thus, it is only able to identify applications that are listening on the specific ports. One could argue that a remote attacker has to deal with the same limitations, but it becomes more complicated if multi-step attacks are considered. When the attacker can first exploit a vulnerability to obtain access to a remote machine with only limited permissions, he is only able to execute a limited set of commands on the system, which could be bound to the permissions of the exploited service. But if the attacker could exploit a locally running service or another local application to elevate his privileges, this possible attack will remain undetected by network based vulnerability scanners. Furthermore, network based vulnerability scanners have a higher probability for an incorrect application detection than local scanners have, as the local scanners can utilize much more information from the target system, such as registry entries, installation logs. The downside of the local scanners is the complexity to integrate

[5] http://sectools.org/tool/amap/

additional clients, since local agents or local scanning applications have to be installed and configured.

However, this difference shows that the detection of vulnerabilities has to be performed comprehensively and extensively to identify all existing vulnerabilities. Thus, vulnerabilities in applications that are running and listening on the network definitely pose a higher risk to the security of the target. Nevertheless, the vulnerabilities of installed applications, which are not executed, should still be considered, as an attacker might be able to start the application to elevate the privileges or maintain the foothold. Additionally, running applications that do not listen to the network might not be exploited remotely, but an attacker could also misuse them. Therefore, the fact that an application is running and listening should be integrated into the scoring of vulnerabilities. One could think about a multiplier that represents the fact if the affected application is running and listening. Hence, an active vulnerability with an application that is running and listening can receive a multiplier of 1, which would mean that the CVSS score will receive the full weight. If the application is running but is not listening on open ports, the exploitation requires a preemptive infiltration of the machine. Thus, the multiplier could be set to 0.8, which would lessen the basic CVSS score since the access is not directly possible. In the last scenario, the passive vulnerability of an unused application has to be activated by starting the application. Then, an attacker has to infiltrate the system and start the application before being able to benefit from the corresponding vulnerability. Therefore, the multiplier could be set to 0.5 as this scenario has the highest requirements that have to be satisfied before an exploitation is possible. The final formula to compute the vulnerability score includes the multiplier and is illustrated in Equation 5.5.

$$FinalScore = Multiplier * CVSSscore \tag{5.5}$$

$$OverallScore = log(\bigcap_{vulnerability \in detectedVulnerabilities} vulnerability.cvss\_score) \tag{5.6}$$

The multiplier is used to adjust the CVSS score as it was described before. Then the overall scale for the score will still be in the interval between 0 and 10, which allows an interoperability with the current CVSS scores.

Finally, severity scores could be assigned to full computers that have been scanned. Therefore, the results for the individual vulnerabilities have to be combined. Several possibilities to combine the scores for the single vulnerabilities were discussed and all methods show different advantages and disadvantages. For example a simple maximum of all the identified vulnerabilities, would still produce results in a suitable scale of 0 to 10. The problem is that the amount of vulnerabilities will not have any effect on the score, which means a system with one vulnerability of a score of 10 will receive a higher overall score than a system with 20 vulnerabilities with a score of 8. Thus, the combination has to incorporate the score of each of the vulnerabilities. Then a multiplication or an addition of all the scores was evaluated, but many test cases result in numerous vulnerabilities, which were discovered. This in turn would produce an overall score, which grows to fast. If several vulnerabilities have a score of 10, the multiplication might result in an overall score in the range of millions or billions. A combination of the multiplication and a logarithm could be a better choice, since the logarithm will drastically decrease the growth of the overall score. The equation for this score is illustrated in Equation 5.6. Even, when numerous vulnerabilities have a score of 10 then the logarithm will reduce the overall score to the number of these vulnerabilities, which is a reasonable measure.

5.5 Conclusion

This chapter discusses the challenges and possible solutions that arise from the necessity of a high data quality. The available information, which is collected from the internet, suffers from unchecked, outdated, or missing properties for the different vulnerabilities. Since the information has to be included into the HPI-VDB, it was necessary to find ways to circumvent the described problems. First of all the defective attribute has to be identified. It is easy to identify missing attributes, since they are simply not available in the specific source. Then a correlation method can be used to enrich the vulnerability with information from other sources. This correlation mechanism utilizes the CVE identifier, which was described in Section 2.3.1. Furthermore, an approach was presented that allows deriving solutions from the textual description of the vulnerability automatically. This solution creation was created, because the information about possible solutions is often missing in several sources. Nevertheless, it is likely that those solutions provide the highest benefit for average users of a vulnerability database, since they include instructions to resolve the vulnerability.

Another aspect of assuring high data quality is the detection of incorrect information for a vulnerability. If the incorrect information is located in an attribute that is used to identify specific properties of the vulnerabilities, a detection of this defect can be achieved by a check against the commonly used standards. The field of vulnerabilities includes several standards to define and identify different properties of the vulnerability, such as CPE, CAPEC, CVSS, and CWE. Thus, those identifiers and schemes should be used as often as possible to allow an easy interoperability and guarantee an error-free identification of specific requirements and impacts for a deep analysis. It is especially important for CPE identifiers, since those CPEs are used to identify applications of a specific version. Therefore, even a minor difference in the CPE would result in an inaccuracy of the analytic approach, since it would assume a different application. Thus, a correction approach was implemented to deal with outdated or incorrectly created CPEs. The approach will correct the flaws automatically and reassign the corresponding vulnerabilities to the correct version of the CPE.

The third approach that was described in this chapter can be used to validate, correct, or create CVSS attributes automatically. Therefore, two different automatic classification methods have been created, which use a naive Bayes and a neural network approach for the classification. The classifiers have been trained on real world data, which was exported from the HPI-VDB. Then, a test dataset and an additional validation dataset was used to perform a thorough evaluation for each approach. The experiments show that an automatic classification of the CVSS attributes is possible, with a promising accuracy of more than 50% for a combination of four different attributes and 108 possible classification results. If only single attributes are considered, the classifier could even achieve up to 90% accuracy. Thus, these approaches can be used to reduce the manual workload of vulnerability experts, who have to investigate the vulnerability reports.

Finally, some thoughts about an adjustment for the scoring mechanism of vulnerabilities conclude the chapter. The difference to the well-established CVSS method is to include passive vulnerabilities and adjust the score accordingly. Overall, this chapter dealt with several ways to increase the quality of attributes for vulnerabilities by enrichment, validation, and correction of various properties. The approaches have been implemented and tested to complete the comprehensive vulnerability model and increase the reliability of that information.

Chapter 6

Future Work and Conclusion

6.1 Conclusion

6.1.1 Vulnerability Detection

This thesis addresses the challenges in utilizing vulnerability information to increase the security of a target IT infrastructure. Therefore, different use cases that could be applied to single systems or large corporate networks have been presented. Thus, both average users and security professionals are able to benefit from the proposed approaches. The importance of a secure environment has been presented and it applies to both of the different target groups. Therefore, the average user has to care about the security of his home network or his computer, which could be used to perform critical actions, such as home automation, storage for documents and personal data, and financial transactions. On the other hand, corporate networks usually contain sensitive information about the company and security issues could lead to several infections at once. Furthermore, modern companies organize their daily work and internal processes with IT systems or completely rely on IT infrastructure to operate their business. Nevertheless, the conditions for those user groups have to be considered, which are tremendously different. These different requirements were also incorporated in the design of the analytic approaches. Hence, a limited version of the vulnerability analysis was designed and implemented, which mainly focuses on the web browser of a user and was presented in [25]. Since the browser is usually the only

application, which interacts with other IT systems in the Internet, it is often regarded as the highest risk for potential attacks. Therefore, it is essential to protect the browser, as it is the potential entrance point for IT criminals. Beside protection mechanisms, such as disabling javascript from unknown sources, the user should ensure that the browser is up-to-date and does not contain vulnerabilities. Since the majority of attacks exploit known vulnerabilities to break into the target system [20] the vulnerabilities of the web browser have to be resolved immediately after disclosure. In the other case, operators of corporate IT systems can directly perform deeper investigations of their systems. Therefore, they can use inventory data, application scanners, or vulnerability scanners to identify potential vulnerabilities in their infrastructure. The results of the application detection can be used to identify correlated vulnerabilities and investigate possible countermeasures. Beyond that, companies already started to collect information about the systems interior status to be able to identify user activities in the corporate network. This information could also be used to identify possible vulnerabilities in the corporate networks and on the user devices as it was described in [25]. Then, the detection results can be presented to the operator, who can investigate the information and ultimately resolve the vulnerabilities. The representation of vulnerability information in large company networks is a complex task by itself. Therefore, attack graphs have been explored, adapted and simplified to allow an easy recognition of the important information, as it was presented in [23]. Additionally, information about solutions of vulnerabilities have been incorporated into the attack graphs to visualize the consequences of applying different countermeasures to the overall infrastructure.

6.1.2 Data Quality

Additional challenges arise with the distributed nature of vulnerability information. Since the information is widespread in several sources, it also varies in the data format. This variance results in different properties for the vulnerabilities and requires a unification. Thus, a comprehensive vulnerability model was created that contains a combination of all relevant attributes. Furthermore, the

model utilizes established standards whenever possible, which allows an interoperability and integration into other security approaches. Since these identifiers are not always in place, a suitable and scalable transformation was required, which was also integrated into the import functionality of the database. However, the available information often prevents an adequate and complete utilization of the comprehensive model. Thus, the minimal requirements for the vulnerability model have been established to avoid information losses. Additionally, different approaches to extend the available information by derivation methods were implemented that allow the creation of different attributes during the import procedure. Beside these extension mechanisms, it was necessary to ensure a high accuracy of the information, since wrong relations between affected applications and vulnerabilities would produce incorrect analysis results. Thus, the before mentioned derivation methods can be used to ensure the correctness of the vulnerability information by applying the derivation and comparing the results to the available values of the characteristic, as it was presented in [24]. Additionally, a reliability measure was included for each of the integrated sources. This reliability is used for a decision when the information differs in the different sources. Furthermore, the employment of the established standards and identifiers have to be ensured, since the analytic approaches do not consider similarities in identifiers, it merely checks for exact matches. Thus, even small inconsistencies have to be corrected, as it was performed with the CPE identifiers.

All of these adjustments have to be performed automatically, because the amount of vulnerability information, which is processed periodically, does not allow a manual interference by security experts. Thus, the different approaches have been designed to work automatically with the help of dictionaries for identifiers or benefit from insights that were gained from machine learning mechanisms.

6.2 Summary of Contributions

This thesis deals with several challenges that arise in the processing of vulnerability information for security analysis. Different fields, such as vulnerability model engineering, information representation, data verification, and classifica-

tion based on machine learning have been touched in this work. The different contributions can be summarized in three main categories, which are composed of different parts.

The first category deals with the retrieval and the quality assurance of the vulnerability information. Therefore, a proposed vulnerability model is used which was presented in [53]. This model was improved and the normalization of the vulnerability information was explained in [22] with a special focus on the scalability and autonomy of the approaches. These normalization steps have been included in the import procedures of the HPI-VDB. Additionally, different methods to ensure the data quality were developed to always guarantee a utilization of established standards and their identifiers. Therefore, correction methods and lookup procedures have been created, such as the correction and lookup of CPE identifiers. Finally, inconsistencies between different sources or the absence of information has to be considered. These challenges were addressed in [24], where missing CVSS attributes, which are fundamental for multiple analytic approaches, are derived from textual information. These derivation methods are based on machine learning approaches that automatically process the information.

The second category addresses the challenge of an appropriate representation of the results. Therefore, the concept of attack graphs is employed. The attack graph creation with MulVAL was developed further and a suitable interface was created to provide vulnerability information for an external tool that creates the attack graph. However, this graph representation is complex and often not adequate. Thus, a simplistic form of the attack graph was integrated into the system, which uses the network structure as the base. It enriches the graphical representation of the network with vulnerability information and highlights vulnerable systems and connections. Furthermore, the automatically created solutions of the vulnerabilities are integrated to allow a direct assessment of the effect of the individual solutions. The user is able to enable the solutions in the graph and will be presented with the resulting scenario, which illustrates the changes that were caused by the application of the solution.

The third category includes the approaches to detect and identify vulnera-

bilities in the specific target environments. Therefore, an automatic detection approach was created that strongly depends on the accurate identification of the inner structure of the environments. Thus, the installed applications have to be identified with a high level of detail that includes the version of the application as well. This automatic detection can be performed with an active scanning approach that examines the target system and identifies installed applications. Thus, the direct detection can be performed on the target system by investigating special paths, entries in the registry or querying the package management. On the other hand, another approach that is less performance intensive was designed for larger corporate networks. The passive detection approach utilizes information from event logs to identify running and communicating applications. These applications can also be classified as more severe, since they have a network connection and are executed. Thus, an attacker might be able to remotely penetrate the network through a vulnerability in one of those applications. Additionally, many large companies meanwhile log events in their environments nonetheless, so this information could directly be used to create an overview of the running applications. Irrespective of the utilized method, the results of the application detection have to be transformed to CPE identifiers, which is the established standard to identify applications. Then those CPE identifiers can be used to perform a lookup in the database and retrieve the related vulnerabilities for the set of applications. Afterwards, the remaining requirements of the vulnerabilities can be evaluated, such as a specific operating system or a required access vector.

All of the explained approaches have been implemented and most of them were also deployed at the publicly available vulnerability database HPI-VDB [27]. Thus, the applicability and practical feasibility can be tested on the website. Furthermore, the different detection mechanisms were also applied on real scenarios to prove the workflow with real-world data.

6.3 Future Work

The work on vulnerability analysis can be continued with additional correlation approaches. Especially, the addition of further sources of vulnerability informa-

tion would result in a more comprehensive collection of vulnerability information. Besides the addition of vulnerabilities, additional sources can be used to verify or complete the information about already existing vulnerabilities. Thus, the data quality could be increased with more sources. A better data quality also results in a higher accuracy of the vulnerability detection, since the knowledge about requirements and impact of all vulnerabilities is fundamental for an appropriate analysis. The foundation for several types of vulnerability sources was created with the reader interface, which has to be implemented for each source. However, the implementation of the RSS reader already solves different challenges and new RSS sources will only require a new regular expression to extract the corresponding fields.

Another point for further work is the creation of additional end-points in the API. The API provides direct access to the vulnerability information and can be used by third-party software to receive vulnerability information for specific use-cases. Thus, the database can serve as a knowledge base for additional security mechanisms. The API can then be queried to receive the vulnerability information with the option of filtering the results. The filtering reduces the necessary data transfer and allows a restriction of the result to the important entries. Currently, two different types for the result have been implemented. Therefore, either the full vulnerability record with all references, solutions, and conditions or a reduced version of the record, which omits the conditions, can be retrieved. For a possible application in the field of threat intelligence, a new type could be created, which includes the conditions, but omits other unnecessary attributes, such as references or timestamps. A confinement for the set of results would also reduce the workload in composing the results.

Additional improvements can be done for attack graphs. As it was elaborated, traditional representations of attack graphs often suffer from high complexity and the resulting problems in interpreting the results. Thus, further work on reducing the complexity should increase the user experience and enhance the usability of attack graphs. One possibility would be an interactive graph that abstracts several details and only visualizes detailed attack steps if the focus of the viewer is centered on the specific part of the graph. Another idea would be

the utilization of graph databases to efficiently allow a traversal of the graph. The current implementation of the simplistic graph is a data structure of nodes and edges with several attributes in the JSON format. The visualization of the graph is done with *Springy.js* [30] and different tests already showed the limitation of the approach if large networks with tens or hundreds of nodes are considered. So one possibility to increase the usability of the attack graph approach would be the integration of another front-end for the visualization of the graph. Moreover, the reasoning and traversal of the simplistic graph are currently performed in Python, which has a large impact on the performance of the approach. Larger scenarios would also benefit from a replacement with a graph database that should have a better performance in traversing large graphs. Finally, these changes could be evaluated by conducting a user study for the attack graph approach. Then, it would be possible to find the most used graph, whether users would prefer the simplistic graph with possible improvements and different visualization front-ends or the traditional graph with MulVAL.

Bibliography

[1] A. Azodi, D. Jaeger, F. Cheng, and C. Meinel. A new approach to building a multi-tier direct access knowledgebase for ids/siem systems. In *Dependable, Autonomic and Secure Computing (DASC), 2013 IEEE 11th International Conference on*, pages 118–123. IEEE, 2013. doi:10.1109/DASC.2013.48.

[2] S. Barnum. Standardizing cyber threat intelligence information with the structured threat information expression (stix). *MITRE Corporation*, 11:1–22, 2012. URL `http://www.standardscoordination.org/sites/default/files/docs/STIX_Whitepaper_v1.1.pdf`.

[3] G. Bartlett, J. Heidemann, and C. Papadopoulos. Understanding passive and active service discovery. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 57–70, New York, NY, USA, 2007. ACM. doi:10.1145/1298306.1298314.

[4] M. Bishop. Vulnerabilities analysis. In *Proceedings of the Recent Advances in intrusion Detection*, pages 125–136, 1999.

[5] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker. Beyond heuristics: learning to classify vulnerabilities and predict exploits. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 105–114. ACM, 2010. doi:10.1145/1835804.1835821.

[6] A. Buttner, T. M. Corporation, N. Ziring, and N. S. Agency. Common platform enumeration (cpe)—specification. *URL http://cpe.mitre.org*, 2008. doi:10.1.1.210.2673.

[7] Carnegie Mellon University. Cert/cc vulnerability report form. `https://vulcoord.cert.org/VulReport/form`, 2017. [accessed 12-March-2018].

[8] Center for Internet Security. Home / oval repository. `https://oval.cisecurity.org/repository/download`, 2018. [accessed 22-March-2018].

[9] F. Cheng, M. Gawron, and C. Meinel. Attack graph construction. `https://hpi.de/meinel/security-tech/security-analytics/attack-graph-construction.html`, 2018. [accessed 13-June-2018].

[10] F. Cheng, S. Roschke, and C. Meinel. An integrated network scanning tool for attack graph construction. In *Proceedings of the 6th International Conference on Grid and Pervasive Computing (GPC'11)*, pages 138–147, 2011. doi:10.1007/978-3-642-20754-9_15.

[11] P. Cichonski, D. Waltermire, and K. Scarfone. Common platform enumeration: Dictionary specification version 2.3. *NIST Interagency/Internal Report (NISTIR)-7697*, 2011. doi:10.6028/NIST.IR.7697.

[12] J. Daiber, M. Jakob, C. Hokamp, and P. N. Mendes. Improving efficiency and accuracy in multilingual entity extraction. In *Proceedings of the 9th International Conference on Semantic Systems*, pages 121–124. ACM, 2013. doi:10.1145/2506182.2506198.

[13] R. Deraison, R. Gula, and T. Hayton. Passive vulnerability scanning: Introduction to nevo. *Revision*, 9(1-13):7, 2003. URL `http://www.vodun.org/papers/net-papers/gula_passive_scanning_tenable.pdf`.

[14] Dimitar Kostadinov. Ghostnet - part i. `https://resources.infosecinstitute.com/ghostnet-part-i/`, 2013. [accessed 13-April-2018].

[15] E. A. Fisch, G. B. White, and U. W. Pooch. Computer system and network security. 1996. doi:10.1201/1079/43236.25.8.19980201/30188.4.

[16] Flexera. Computer security research - secunia. `https://secuniaresearch.flexerasoftware.com/community/research/`, 2018. [accessed 22-March-2018].

[17] E. Frank, G. W. Paynter, I. H. Witten, C. Gutwin, and C. G. Nevill-Manning. Domain-specific keyphrase extraction. In *16th International Joint Conference on Artificial Intelligence (IJCAI 99)*, volume 2, pages 668–673. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999. URL `https://hdl.handle.net/10289/1508`.

[18] J. Franklin, C. Wergin, and H. Booth. Cvss implementation guideline. National Institute of Technology, 2014. doi:10.6028/NIST.IR.7946.

[19] S. Frei, T. Duebendorfer, and B. Plattner. Firefox (in) security update dynamics exposed. *ACM SIGCOMM Computer Communication Review*, 39(1):16–22, 2008. doi:10.1145/1496091.1496094.

[20] Gartner Inc. Focus on the biggest security threats, not the most publicized. `"https://www.gartner.com/smarterwithgartner/focus-on-the-biggest-security-threats-not-the-most-publicized`, 2017. [accessed 10-March-2018].

[21] Gartner Inc. Magic quadrant for security information and event management. `https://www.gartner.com/doc/reprints?id=1-4JMUB31&ct=171031&st=sb`, 2017. [accessed 14-June-2018].

[22] M. Gawron, F. Cheng, and C. Meinel. Automatic detection of vulnerabilities for advanced security analytics. In *Proceedings of the 17th Asia-Pacific Network Operations and Management Symposium (APNOMS'15)*, pages 471–474. IEEE, 2015. doi:10.1109/APNOMS.2015.7275369.

[23] M. Gawron, F. Cheng, and C. Meinel. Automatic vulnerability detection for weakness visualization and advisory creation. In *Proceedings of the 8th International Conference on Security of Information and Networks (SIN'15)*, pages 229–236. ACM Press, 2015. doi:10.1145/2799979.2799986.

[24] M. Gawron, F. Cheng, and C. Meinel. Automatic vulnerability classification using machine learning. In *International Conference on Risks and Security of Internet and Systems*, pages 3–17. Springer, Springer, 2017. doi:10.1007/978-3-319-76687-4_1.

[25] M. Gawron, F. Cheng, and C. Meinel. PVD: Passive vulnerability detection. In *Information and Communication Systems (ICICS), 2017 8th International Conference on*, pages 322–327. IEEE, Apr. 2017. doi:10.1109/iacs.2017.7921992.

[26] R. Gula. Passive vulnerability detection. *Network Security Wizards*, 9:7, 1999. URL `http://www.vodun.org/papers/net-papers/gula_passive_vulnerability_detection.pdf`.

[27] Hasso Plattner Institute. HPI Vulnerability Database. `https://hpi-vdb.de/`, 2018. [accessed 26-March-2018].

[28] Hasso Plattner Institute. Real-time Event Analysis and Monitoring System (REAMS). 2018. URL `https://hpi.de/en/meinel/security-tech/security-analytics/reams.html`. [accessed 12-March-2018].

[29] D. Hein and H. Saiedian. Predicting attack prone software components using repository mined change metrics. In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*, pages 554–563, 2016. doi:10.5220/0005812905540563.

[30] D. Hotson. Springy - a force directed graph layout algorithm in javascript, 2013. URL `http://getspringy.com/`. [accessed 05-July-2018].

[31] D. Jaeger, A. Azodi, F. Cheng, and C. Meinel. Normalizing security events

with a hierarchical knowledge base. In *Information Security Theory and Practice*, pages 237–248. Springer, 2015. doi:10.1007/978-3-319-24018-3_15.

[32] S. Jajodia, S. Noel, and B. O'Berry. Topological analysis of network attack vulnerability. In *Managing Cyber Threats*, pages 247–266. Springer, 2005. doi:10.1007/0-387-24230-9_9.

[33] Jonathan Crowe. Must-know phishing statistics 2017. `https://blog.barkly.com/phishing-statistics-2017`, 2017. [accessed 12-March-2018].

[34] Kevin McCaney. Hack of rsa's secureid exploited flash vulnerability. `https://gcn.com/articles/2011/04/04/rsa-hack-securid-adobe-flash.aspx`, 2011. [accessed 13-April-2018].

[35] G. Lyon. Nmap: the network mapper - free security scanner. `https://nmap.org/`, 2011. [accessed 13-June-2018].

[36] McAfee. Economic impact of cybercrime - no slowing down. `https://www.mcafee.com/enterprise/en-us/assets/reports/restricted/economic-impact-cybercrime.pdf`, 2018. [accessed 12-March-2018].

[37] P. Mell, K. Scarfone, and S. Romanosky. Common vulnerability scoring system. *Security & Privacy, IEEE*, 4(6):85–89, 2006. doi:10.1109/MSP.2006.145.

[38] M. S. Merkow and J. Breithaupt. *Information security: Principles and practices*. Pearson Education, 2014. URL `http://cds.cern.ch/record/1749101`.

[39] Microsoft. Windows remote management. `https://msdn.microsoft.com/en-us/library/aa384426.aspx`, 2016. [accessed 12-March-2018].

[40] Microsoft. Microsoft security bulletins. "https://technet.microsoft.com/en-us/security/bulletins.aspx, 2018. [accessed 24-March-2018].

[41] Mitre Corporation. Oval - oval interpreter. http://oval.mitre.org/language/interpreter.html, 2012. [accessed 22-March-2018].

[42] Mitre Corporation. Cve list main page, 2018. URL http://cve.mitre.org/cve/index.html. [accessed 22-June-2018].

[43] Mitre Corporation. The mitre corporation, 2018. URL https://www.mitre.org/. [accessed 22-June-2018].

[44] Mozilla. Security advisories for firefox. https://www.mozilla.org/en-US/security/known-vulnerabilities/firefox/, 2016. [accessed 11-March-2018].

[45] National Institute of Standards and Technology. Cwe - common weakness enumeration. http://nvd.nist.gov/cwe.cfm, 2016. [accessed 14-March-2018].

[46] National Institute of Standards and Technology. National vulnerability database. http://nvd.nist.gov/, 2017. [accessed 22-April-2018].

[47] Network Working Group. Rfc 5424: The syslog protocol. https://tools.ietf.org/html/rfc5424, 2009. doi:10.17487/RFC5424. [accessed 11-February-2016].

[48] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller. Predicting vulnerable software components. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 529–540. ACM, 2007. doi:10.1145/1315245.1315311.

[49] Offensive Security. Exploits database by offensive security. https://www.exploit-db.com/, 2017. [accessed 22-June-2018].

[50] OSVDB Project Group. Osvdb: Open sourced vulnerability database. `http://www.osvdb.org`, 2017. [accessed 28-March-2018].

[51] X. Ou, S. Govindavajhala, and A. W. Appel. Mulval: A logic-based network security analyzer. In *USENIX Security Symposium*, volume 8, pages 113–128. Baltimore, MD, 2005. URL `http://dl.acm.org/citation.cfm?id=1251398.1251406`.

[52] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL `http://www.aclweb.org/anthology/D14-1162`.

[53] S. Roschke, F. Cheng, R. Schuppenies, and C. Meinel. Towards unifying vulnerability information for attack graph construction. In *Proceedings of the 12th Information Security Conference (ISC'2009)*, pages 218–233, 2009. doi:10.1007/978-3-642-04474-8_18.

[54] S. Russell, P. Norvig, and A. Intelligence. *A modern approach*, volume 25. Citeseer, 1995.

[55] A. Sapegin, D. Jaeger, A. Azodi, M. Gawron, F. Cheng, and C. Meinel. Hierarchical object log format for normalisation of security events. In *Proceedings of the 9th International Conference on Information Assurance and Security (IAS 2013)*, Tunis, Tunisia, 12 2013. IEEE CS.

[56] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen. Predicting vulnerable software components via text mining. *IEEE Transactions on Software Engineering*, 40(10):993–1006, 2014. doi:10.1109/TSE.2014.2340398.

[57] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3):297–336, 1999. doi:10.1023/A:1007614523901.

[58] M. Schumacher, C. Haul, M. Hurler, and A. Buchmann. Data mining in vulnerability databases. *In 7. Workshop "Sicherheit in vernetzten Systemen" (Hamburg, Germay, March 2000), DFN-CERT*, 12, 2000.

[59] SecurityFocus. Securityfocus. `https://www.securityfocus.com/bid/`, 2018. [accessed 24-March-2018].

[60] R. W. Shirey. Internet security glossary. *Request For Comment (RFC)*, 2828. doi:10.17487/RFC2828.

[61] R. W. Shirey. Internet security glossary, version 2 (2007). *Request For Comment (RFC)*, 4949. doi:10.17487/RFC4949.

[62] Tenable Inc. Plugins — tenable. `https://www.tenable.com/plugins`, 2018. [accessed 13-June-2018].

[63] Tenable Network Security. Nessus vulnerability scanner. `https://www.tenable.com/products/nessus-vulnerability-scanner`, 2016. [accessed 13-June-2018].

[64] Text Fixer. Common English Words List. `http://www.textfixer.com/tutorials/common-english-words.txt`, 2017. [accessed 11-March-2018].

[65] The MITRE Corporation. Capec - common attack pattern enumeration and classification (capec). `https://capec.mitre.org/`, 2017. [accessed 22-April-2017].

[66] Vikram Thakur. The sykipot attacks. `https://www.symantec.com/connect/blogs/sykipot-attacks`, 2011. [accessed 13-April-2018].

[67] w3schools. Browser statistics. `http://www.w3schools.com/browsers/browsers_stats.asp`, 2015. [accessed 11-March-2018].

[68] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012. doi:10.1017/S1471068411000494.

[69] D. Wijayasekara, M. Manic, and M. McQueen. Vulnerability identification and classification via text mining bug databases. In *IECON 2014-40th Annual Conference of the IEEE Industrial Electronics Society*, pages 3612–3618. IEEE, 2014. doi:10.1109/IECON.2014.7049035.

[70] Zero Day Initiative. Zdi: Published advisories. `https://www.zerodayinitiative.com/rss/published/`, 2018. [accessed 13-April-2018].

[71] S. Zhang, X. Ou, and D. Caragea. Predicting cyber risks through national vulnerability database. *Information Security Journal: A Global Perspective*, 24(4-6):194–206, 2015. doi:10.1080/19393555.2015.1111961.