Hochschule Konstanz
Department of Computer Science

Submitted by
Tom Kosacki
Student Number [redacted]

[redacted]

Bachelor Thesis

Simplifying Vulnerability-Scan Results

Konstanz, 11th July 2023

Hochschule Konstanz
Department of Computer Science

Bachelor Thesis

Simplifying Vulnerability-Scan Results

by

Tom Kosacki

in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science
in Applied Computer Science

at the Hochschule Konstanz University of Applied Sciences,

Student Number: [redacted]

Date of Submission: 11th July 2023

Supervisor: **Prof. Dr. Hanno Langweg**
Second Examiner: **Prof. Dr. Dirk Staehle**

Hiermit erkläre ich, Tom Kosacki, geboren am [redacted]

(1) dass ich meine Bachelorarbeit mit dem Titel:

„Simplifying Vulnerability-Scan Results“

in der Informatik unter Anleitung von Professor Prof. Dr. Hanno Langweg selbständig und ohne fremde Hilfe angefertigt habe und keine anderen als die angeführten Hilfen benutzt habe;

(2) dass ich die Übernahme wörtlicher Zitate, von Tabellen, Zeichnungen, Bildern und Programmen aus der Literatur oder anderen Quellen (Internet) sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

(3) dass die eingereichten Abgabe-Exemplare in Papierform und im PDF-Format vollständig übereinstimmen.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

[redacted]

Konstanz den 11.07.2023

Abstract

This thesis emphasizes problems that reports generated by vulnerability scanners impose on the process of vulnerability management, which are a. an overwhelming amount of data and b. an insufficient prioritization of the scan results.

To assist the process of developing means to counteract those problems and to allow for quantitative evaluation of their solutions, two metrics are proposed for their effectiveness and efficiency. These metrics imply a focus on higher severity vulnerabilities and can be applied to any simplification process of vulnerability scan results, given it relies on a severity score and time of remediation estimation for each vulnerability.

A *priority score* is introduced which aims to improve the widely used Common Vulnerability Scoring System (CVSS) base score of each vulnerability dependent on a vulnerability's ease of exploit, estimated probability of exploitation and probability of its existence.

Patterns within the reports generated by the Open Vulnerability Assessment System (OpenVAS) vulnerability scanner between vulnerabilities are discovered which identify criteria by which they can be categorized from a remediation actor standpoint. These categories lay the groundwork of a final simplified report and consist of updates that need to be installed on a host, severe vulnerabilities, vulnerabilities that occur on multiple hosts and vulnerabilities that will take a lot of time for remediation. The highest potential time savings are found to exist within frequently occurring vulnerabilities, minor- and major suggested updates.

Processing of the results provided by the vulnerability scanner and creation of the report is realized in the form of a python script. The resulting reports are short, straight to the point and provide a top down remediation process which should theoretically allow to minimize the institutions attack surface as fast as possible. Evaluation of the practicality must follow as the reports are yet to be introduced into the Information Security Management Lifecycle.

Contents

List of Figures

List of Tables

List of Source Codes

1

Introduction

Institutions of Higher Education (IHEs) are increasingly reliant on Information Technology (IT) systems to support their teachings, research, and administration. However, with this increasing reliance and the increasing willingness of threat actors to compromise IHEs comes the risk of data loss and inability to operate. To mitigate these risks, vulnerability management has become a critical component of IT security for IHEs. Automated vulnerability scanning is a common approach used to identify vulnerabilities in IT systems, but the volume of vulnerabilities discovered can become overwhelming with increasing institution- and thus network-size, making evaluation and remediation complex and resource-intensive tasks.

In this Bachelor's thesis the issue of vulnerability prioritization and report simplification for IHEs are explored. Automated vulnerability scans are utilized to identify vulnerabilities and a focus on optimizing the scan reports and the prioritization of vulnerabilities to reduce the complexity of remediation is set.

The objective of this thesis is to provide practical guidance and tools for IHEs to simplify existing vulnerability scan results using various techniques throughout the scanning process. These tools include bash and python scripts as well as Structured Query Language (SQL) queries. The thesis reviews existing literature on vulnerability scan prioritization, modifies findings, if applicable, and presents a case study on the use of vulnerability prioritization in a higher education institution.

Before developing means to simplify vulnerability scan results, four important questions must be addressed first:

- What is the current complexity of vulnerability scan results?
- How do IHEs handle vulnerability scan results?
- How resource intensive is the remediation process that comes with the scan results?
- What is considered a useful prioritization method?

Overall, this Bachelor's thesis aims to contribute to the improvement of vulnerability management in IHEs by providing a framework for vulnerability scan report simplification that is tailored to the resource constraints that exist within IHEs.

1.1. Related Works

No previous academic work has been found that focused on minimizing the amount of information that comes from automated vulnerability scan reports, but the necessity for a simplification of vulnerability scan results has been noted several times [Har+18; Alp+19]. Many systems for the prioritization of vulnerabilities exist with the CVSS being the global standard. This standard does not perform well enough in real world scenarios when it comes to vulnerability prioritization [Rey+22]. A multitude of academic research has been conducted to find universally applicable vulnerability prioritization methods. Some of which focused solely on the importance of remediation of certain vulnerabilities [Jac+23; Alm+17; Rey+22]. These systems provide exploit probability estimations for each vulnerability that perform better compared to the CVSS, but still lack context. Others explored prioritization formulas that integrate further domain specific dimensions [Sha+22]. This particular research concluded with no time savings and no time-to-remediation improvements compared to a top down remediation process.

In general, although numerous studies have identified effective methods for prioritizing significant vulnerabilities, none have effectively mitigated the effort required to address an extensive vulnerability report.

1.2. Structure

This thesis is structured into four main chapters, each addressing a specific aspect of vulnerability scanning in Institutions of Higher Education (IHEs).

Chapter 2 provides a short introduction to vulnerability scanning, encompassing the various types of vulnerability scans and their significance in IHEs. This section aims to establish a foundation for understanding the purpose and relevance of vulnerability scanning within the context of IHEs.

Chapter 3 focuses on the evaluation of the complexity of the reports automatically generated from vulnerability scans. By evaluating these reports, challenges and issues that hinder the remediation process are identified. This evaluation serves as a basis for recognizing the need for improvements in the prioritization of

vulnerabilities and the simplification of scan reports.

Building upon the previous findings, chapter 4 focuses on developing means to prioritize vulnerabilities and reduce the complexity of scan reports. By employing effective means of prioritization, the aim is to enhance the efficiency and effectiveness of the remediation process.

Finally, in chapter 5, the developed methods and techniques for prioritization and simplification are implemented in a self-contained script. Challenges and findings during implementation are documented in this section.

Case Study This thesis is accompanied by a case study, with the HTWG Konstanz as an example. For information security reasons, no specific information about the scanned systems is provided. Relevant key data about the scans performed are:

1. the HTWG network has a /16 subnet.
2. the university is divided into 20 administrative units for the purpose of this thesis.
3. scans were performed from outside as well as from inside the university.

1.3. Technical Terms and Tools used for Research

(Greenbone) OpenVAS

OpenVAS is a feature-rich vulnerability scanner that supports both authenticated and unauthenticated testing. It offers extensive protocol coverage, performance tuning options, and a powerful internal programming language for custom vulnerability tests. OpenVAS relies on a regularly updated feed to provide a wide range of tests, ensuring accurate and up-to-date vulnerability detection [Grend].

Nmap

”Nmap (’Network Mapper’) is a free and open source utility for network discovery and security auditing.”[Lyond]

CVSS

The CVSS is a standardized framework used to assess and communicate the severity and impact of security vulnerabilities. It provides a score that represents the relative severity of a vulnerability, helping organizations prioritize their remediation efforts.

The CVSS score is based on several metrics that evaluate different aspects of a vulnerability, including its impact on confidentiality, integrity, and availability of a system, as well as the complexity and exploitability of the vulnerability. These metrics are divided into three groups: the Base metrics, Environmental metrics and the Temporal metrics. ”The NVD does not currently provide ’temporal scores’ (metrics that change over time due to events external to the vulnerability) or ’environmental scores’ (scores customized to reflect the impact of the vulnerability on your organization).”[NISnd]

SQLite

”SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform”[Connd].

DB Browser for SQLite

The DB Browser allows to visually interact with SQLite databases and can be used as a GUI for the SQLite CLI.

Exploit-DB

”The Exploit Database is a CVE compliant archive of public exploits and corresponding vulnerable software, developed for use by penetration testers and vulnerability researchers.”[Offnd] While Exploit-DB is intended for the use of testers and researchers, it can easily be used by threat actors as well.

List of Acronyms

CTE Common Table Expression

CVE Common Vulnerability and Exposure

CVSS Common Vulnerability Scoring System

CWE Common Weakness Enumeration

DMZ Demilitarized Zone

DNS Domain Name Server

DoS Denial of Service

DoS Denial of Service

EOL End of Life

EPSS Exploit Prediction Scoring System

FiRST Forum of Incident Response and Security Teams

GPL General Purpose Language

HPT High Priority Threshold

IANA Internet Assigned Numbers Authority

IHE Institution of Higher Education

IP Internet Protocol

ISMS Information Security Management Systems

IT Information Technology

NVD National Vulnerability Database

OpenVAS Open Vulnerability Assessment System

QoD Quality of Detection

RCE Remote Code Execution

RMS Root Mean Square

SHAP SHapley Additive exPlanations

SQL Structured Query Language

TCP Transmission Control Protocol

WST Wide Spread Threshold

2

Vulnerability Scanning

This chapter will explain what vulnerability scanning is and how it works. The different types of vulnerability scans that can be performed will be covered and the importance of vulnerability scanning for IHEs explained.

2.1. Vulnerability Scanning: An Introduction

According to [Sca+08], vulnerability scanning goes beyond network port and service identification by aiming to identify vulnerabilities instead of relying solely on human interpretation of scanning results. In addition to identifying hosts and their attributes, vulnerability scanners often integrate with network discovery and port/service identification tools, reducing the workload required for comprehensive scanning. Moreover, certain scanners are capable of conducting their own network discovery and port/service identification. The primary objective of vulnerability scanning is to detect various security weaknesses, including outdated software versions, missing patches, and misconfigurations. It also plays a crucial role in verifying compliance with an organization's security policies or identifying deviations from those policies. This is achieved by examining the operating systems and major software applications installed on the scanned hosts and comparing them against information on known vulnerabilities stored in the scanner's vulnerability databases.

Vulnerability scanning is an essential component of proactive cyber security practices aimed at identifying potential weaknesses in computer systems, networks, and applications. By conducting systematic scans, institutions can identify vulnerabilities before they can be exploited by malicious actors. Vulnerability scanning helps institutions assess their security posture, prioritize remediation efforts, and reduce the risk of cyber attacks.

2.2. Types of Vulnerability Scanning

One important aspect to consider is the distinction between active and passive vulnerability scanning. Active scanning involves actively probing systems and networks to identify vulnerabilities, while passive scanning relies on monitoring network traffic and analyzing system logs to identify intrusions and potential vulnerabilities [Wag22]. Both approaches have their own strengths and limitations, and institutions may choose to employ a combination of active and passive scanning techniques based on their specific requirements. In this thesis solely active scanning will be covered.

It is worth noting that vulnerability scanning should be performed regularly and consistently, as new vulnerabilities emerge on a regular basis due to software updates, changes in configurations, or the discovery of previously unknown vulnerabilities.

There are various vulnerability scanning tools and techniques available to organizations for conducting effective vulnerability assessments. Vulnerability scanners usually come in two financial variants:

Commercial Vulnerability Scanners Commercial vulnerability scanning tools, such as Nessus, Qualys and Rapid7 Nexpose[OWA20], are widely used in the industry. These tools offer comprehensive scanning capabilities, extensive vulnerability databases, and advanced reporting features. They can scan networks, systems, and applications for known vulnerabilities, misconfigurations, and compliance issues. Commercial scanners often provide regular updates to their vulnerability databases to keep pace with emerging threats.

Open Source Vulnerability Scanners Open source vulnerability scanning tools,

such as OpenVAS, Nikto, and OWASP ZAP[OWA20], are freely available and offer similar functionality to commercial tools. Some companies (e.g. Greenbone) offer free and commercial solutions in which case the free solution receives intentional drawbacks, i.e. smaller Internet Protocol (IP) batches or less frequent database updates. Other open source scanners often have active developer communities, allowing for community-driven updates and improvements.

Within those variants four different of vulnerability scanning exist which can either be utilized each one on their own, or combined as described in the previous section (2.1). These methods are:

Network Scanners Network scanners focus on identifying vulnerabilities in network devices, such as routers, switches, and firewalls.[Lin04]

Host-based Scanners Host-based scanners focus on vulnerabilities present within individual hosts, such as servers and workstations. These tools assess the operating system, installed software, and configuration settings for known vulnerabilities and weaknesses.[Lin04]

Web Application Scanners Web application scanners specifically target vulnerabilities in web applications. These tools simulate attacks and assess the security posture of web applications by examining the code, inputs, and responses.[MK15]

Manual Techniques In addition to automated scanning tools, manual techniques are often employed to supplement vulnerability scanning efforts. Manual techniques involve in-depth analysis, verification, and penetration testing conducted by skilled security professionals. This approach is usually very time consuming and expensive.

These tools should be aligned with the institution's specific needs and resources. Factors such as the size and complexity of the IT infrastructure, available budget, and level of expertise within the institution influence the selection of suitable scanning tools and the allocation of time and resources for scanning activities.

In the case of this thesis the open source vulnerability scanner OpenVAS which is maintained by Greenbone has been chosen, because it combines network, host-based and web application scanning with little manual input needed in one tool.

2.3. Importance to Institutions of Higher Education

In Microsoft's report, "Microsoft Security Intelligence", 7,854,166 devices out of 9,794,732, or approximately 80% of all reported malware affected devices, were reported from the educational sector [Micnd]. In the years 2021 and 2022 alone, KonBriefing recorded hundreds of successful cyberattacks against international higher education institutions [Konnd]. There are several reasons why the educational sector, especially IHEs, has become the target of cyberattacks on a continual basis. IHEs accumulate lots of sensible personal information including, but not limited to, names, addresses, phone numbers, birth dates, driver's licenses, financial and medical data [Krs23]. This data is highly valuable in the 21st century and is sometimes referred to as "the new 'oil'"[Sch+11]. Secondly, many IHEs conduct cutting-edge research that can be very expensive and labor intensive [Har+18]. The data connected to this research can be of inestimable value. Many IHEs work with research partners in both the public and private sectors. They cannot afford to have the intellectual property they are charged with collecting and protecting breached or attacked [Krs23]. With these valuable resources concentrated in one sector, it is of the utter most importance to detect possible attack vectors before an adversary does. While extensive Information Security Management Systems (ISMS) protocols and IT-Grundschutz policies can provide a good foundation for the IT security of an IHEs, only active evaluation of the institution's systems can provide an overview of the actual security and can be used as a starting point for its hardening.

A significant factor that contributes to the importance of vulnerability scanning in IHEs is their minimalist approach to access restrictions. With minimal effort, it is relatively easy to gain local access to an IHE's network, thereby providing the opportunity to exploit a multitude of vulnerabilities that would otherwise remain inaccessible. Additionally, the number of attack vectors through physical means [Kro13] significantly increases in comparison to private enterprises or institutions, where access to local facilities is typically subject to certain restrictions.

Since IHEs do not have the budget to consult external IT security providers or to perform manual penetration testing on their systems, automated vulnerability scans are the only way to make sense of the security situation within the institutions.

3

Complexity of Remediation

In this chapter the challenges that institutions of higher education face when it comes to remediating vulnerabilities will be discussed. Explanation on how the complexity of the vulnerability scans can be a barrier to effective vulnerability management and how prioritization can help to minimize the complexity of remediation will be given.

The extensiveness of vulnerability scans in the context of IHEs are explored by inferring insights from a larger-scale study and applying these findings to a specific institution. These findings are then waged against the actual report length for a specific IHE. The chapter aims to predict the extent of vulnerability scan report evaluation and to discover potential optimization headroom of remediation efforts accordingly.

3.1. Estimating Vulnerability Scan Report Length

Research found that out of 272 IHEs with a total of 122,360 exposed unique devices 48,218 devices had at least one vulnerability, out of which 31,567 devices had informational vulnerabilities and 16,851 had vulnerabilities with CVSS scores assigned to them, that would make 450 exposed unique devices and 177 devices with at least one vulnerability per IHE. Furthermore a total of 307,332 vulnerabilities

were found on the 48,218 devices, 46,927 had CVSS scores assigned, "Specifically, 4,143 were 'Critical,' 12,452 were 'High,' 26,519 were 'Medium,' 3,813 were 'Low,' and 260,405 were 'Informational'"[Har+18]. From these findings we could assume that, with full network access, the number of vulnerable devices would grow significantly, thus the vulnerability count would rise too. Since the 122,360 exposed unique devices only made up 4.21% of potential devices and scanning the /16 subnet from outside of the HTWG network results in 47,758 alive hosts and 5,004 Domain Name Server (DNS) resolved hosts out of 65,534 potential devices evaluating to 13.1% device exposure.[1] The expected number of vulnerable devices in said IHE increases to 550 $((48,218/272) \cdot (13.1/4.21))$ and the amount of CVSS vulnerabilities proportional to that to 535 $((46,927/48,218) \cdot 550)$ and that of informational vulnerabilities to 2,970 $((260,405/48,218) \cdot 550)$ making up a total of 3,505 vulnerabilities to report.
Each report of an informational vulnerability takes up half a DIN A4 page, while reports of a CVSS scored vulnerability can vary greatly in size and extent, but usually span over one DIN A4 page. This would lead to an overall estimated report length of about 2020 pages.

3.2. Actual Vulnerability Scan Report Length

When running an OpenVAS *Full and Fast* scan from inside the HTWG network, as described in section 5.1, which includes up to 65,534 potential hosts, a total of 1,004 hosts were discovered in the institution. 23,688 total results were collected which were automatically filtered based on a Quality of Detection (QoD) value of 70%. 16.649 of the results were informational (logs) and 2,461 had a CVSS score assigned to them. The exported reports for the 1,004 hosts consists of just over 5,837 pages. When filtering out informational vulnerabilities the total size of the reports shrank to 4,028 pages. Each vulnerability has 2.39 reference URLs and 1.52 Common Vulnerabilities and Exposures (CVEs) assigned to them on average.

Considering an average reading time of 15 minutes per vulnerability, which encompasses comprehending the vulnerability description, examining references,

[1] These are the numbers reported by a network scan with Nmap from outside the institution. Many hosts have later been determined offline, this was caused by firewall protection mechanisms to slow down adversaries.

and assessing the urgency of remediation, the evaluation of the report for the entire institution would demand 615.25 hours. In other terms, this equates to approximately 77 full workdays, assuming an eight-hour workday. It is important to note that this estimation solely pertains to the evaluation phase and does not account for any remediation efforts.

When performing authenticated vulnerability scans, the number of results would increase by a factor of approximately 10 [Hol+11] which would cause the generated report to exceed 40,000 pages.

3.3. Comparison and Handling of Scan Results

The large difference between the estimated number of vulnerabilities, 3,505, and actual number of vulnerabilities discovered, 23,688, was expected and is explained in the following as well as the response of IHEs to the length of the reports.

Comparison The procedure utilized by Harrel et al. in [Har+18] relied on external vulnerability scans. Since each IHE is, or should be, protected from the internet by firewalls, many requests to their underlying hosts would be blocked. Firewall evasion is possible, but it requires individual scan configurations for each institution and maybe even host. Evasion techniques that can be applied to a multitude of firewalls, such as package fragmentation, slow down the scan process significantly. The research performed scans on 272 IHEs which renders both approaches almost impossible. Thereby only those hosts and especially specific ports that were allowed through the firewall were probably scanned in the scope of the research. Additionally some firewalls report scanned hosts which are offline as online to impede network scans. Losing out on hosts would have been compensated by the calculations performed in section 3.1, but a reduction in scannable ports was and could not realistically been taken into account since the number of open ports varies greatly between hosts and the firewall configurations regarding allowed ports can vary as well.

The actual length of the report was not calculated but observed from vulnerability scans performed from within the IHE's Demilitarized Zones (DMZs) and thereby circumvented part of the firewall filtering. This allowed for a larger cov-

erage in hosts and ports as well as less ”noise” of wrongly reported hosts.

The estimation provides a general overview of an institutions potential attack surface which could be discovered by adversaries performing large scale attacks, while the actual confirmed report is important to vulnerability management and provides an overview of what targeted attacks would find out about an institution.

Handling of Scan Results As deducted in section 3.2 the evaluation alone of an automatically generated vulnerability report amounts to about 77 days. With an average of 21 workdays per month the evaluation would require 4 full time employees to work on only this report every month, as it is recommended to perform an automated vulnerability scan once every month [Wal23]. This workload is simply not manageable by IHEs, i.e. Baden-Württemberg provides 58 IT security positions to a total of 32 IHEs averaging to just under 2 positions per IHE [Bad20]. The provided workforce is only half of what would be required for continuous information security management, when expecting the employees to only work on the evaluation of scan reports eight hours every day. Vulnerability scans are being performed, but their resulting report omitted.

The IT security means provided to IHEs are not sufficient for thorough information security management. Since these means are unlikely to change, reduction of the workload is required.

4

Simplification & Prioritization

This chapter will cover the concept of vulnerability prioritization and the different approaches that can be used to simplify scan results. It will also discuss the criteria that can be used to prioritize vulnerabilities, such as the severity of the vulnerability, the likelihood of the vulnerability being exploited and the potential impact of a successful attack.

4.1. Defining Usefulness of Prioritization Methods

Useful prioritization methods are those that effectively reduce the time needed for remediation of likely to be exploited vulnerabilities. This can be achieved by discovering services spread across the network that might require external expertise, services and hosts for which updates are disabled, IP address ranges or domains which pile up relatively large amounts of vulnerabilities and actions that can be taken by the IT administration in order to mitigate wide spread issues. The goal is not to develop an in-depth vulnerability exploit prediction system, which has been done several times before [Jac+23], but a tool that maximizes the cost-benefit relation by minimizing the administrative overhead that comes from overly extensive vulnerability reports and thereby concede valuable time to the actual remediation procedures. Prioritization methods that could fulfill the aforementioned criteria,

but the cost or time for implementation of which would be beyond the resources available to this thesis, will not be considered useful. Methods that lie within this exclusion are for example machine learning approaches or manual report analysis.

To consolidate the usefulness for prioritization methods, a quantitative proof is required. The straight forward approach of using a simple CVSS score per hour, as $\frac{s_{cvss}}{t_{hours}}$ is not sufficient for an important reason. The CVSS score scala is not linear. This means ten vulnerabilities with a CVSS score of 1.0 are not as severe as one vulnerability with a CVSS score of 10.0. With the CVSS score per hour approach, and when assuming the former take 1.0 hours each and the latter takes 10.0 hours, prioritizing either the less severe vulnerabilities or the severe one would result in the same CVSS score per hour of 1.0 even though remediating the severe vulnerability would lead to a much greater security improvement. This changes once the Root Mean Square (RMS) of the CVSS scores is calculated, which will set higher CVSS scores apart from lower ones.

Considering the aforementioned factors, the equation 4.1 has been formulated to calculate the efficiency E in CVSS per hour. The equation incorporates parameters such as V, which represents the set of vulnerabilities with a specific CVSS score s and associated time cost t.

$$E = \frac{\sqrt{\sum_{s,t}^{V} \frac{s^2}{t}}}{|R|} \tag{4.1}$$

Additionally, the parameter R represents the set of vulnerabilities selected for remediation. In the original report, R is equivalent to V, resulting in $|R| = |V|$ when applying the formula. However, when modifications are made to the initial report, such as simplification, it is necessary to consider that $R \subseteq \mathcal{P}(V)$ applies, since each remediation task addresses a set of vulnerabilities. In such cases, it is desirable for $|R| \ll |V|$ to hold.

Another significant metric to consider is the post-remediation count of high, medium, and low severity vulnerabilities. The evaluation of the effectiveness of the remediation effort is represented by an *effectiveness score* denoted as S, which ranges between 0 and 1. A score of 1 signifies the highest level of effectiveness, while a score of 0 indicates the lowest. It is crucial to prioritize the remediation of high severity vulnerabilities over medium and low severity vulnerabilities. Formula 4.2 provides a calculation for the *effectiveness score* while incorporating this

prioritization criterion. The formula requires input values for the count of high severity vulnerabilities (H), medium severity vulnerabilities (M), and low severity vulnerabilities (L) both before and after the simplification process.

$$S = \frac{6}{\frac{H_b}{H_a} \cdot 3 + \frac{M_b}{M_a} \cdot 2 + \frac{L_b}{L_a}} \tag{4.2}$$

Before simplification the *effectiveness score* should always be 1, afterwards a score close to 1 is desirable.

4.2. Result Prioritization

As deducted in section 3 the time needed for result evaluation is way too high for the IT security department of an IHE. Since the evaluation of severity that is provided by the vulnerability scanner in use, which uses the CVSS scoring, does not provide sufficient information for vulnerability management [Spr+21], means of improving the prioritization of vulnerabilities required. Fortunately research has already been conducted in this field. The approach used in this thesis borrows some of those research's findings and combines them not to replace, but to improve the existing CVSS scores to emphasize existing vulnerabilities with high potential impact on the IHE.

Every vulnerability scanner ranks the discovered vulnerabilities by a predefined scoring system of some sort. Most of them, such as OpenVAS, rely on the CVSS rating. These scoring systems provide a general overview for every individual vulnerability. Most of them calculate the score based on multiple factors, e.g. the proximity needed to make use of the vulnerability, the authentication status needed by the attacker, to which extent integrity, confidentiality and availability could be compromised, etc. When the resources to remediate every single vulnerability exist, these scores provide a great sense of where to start with remediation. But this is not the case for IHEs, which consist of thousands of potential targets with very little resources for remediation. In this case the scoring systems do not take enough domain specific vectors into account. Therefore new vectors have to be introduced and their efficiency will later evaluated.

1. **Priority Score** - a value derived from the CVSS base score, estimating a

vulnerability's remediation priority.

2. **Complexity** - a value that estimates the complexity of remediation for each vulnerability.

Combinations of these vectors and information gathered by the vulnerability scanner are then used to generate a comprehensive prioritization overview for the system administrators.

Some vectors frequently discussed in vulnerability remediation efforts were excluded, e.g. available work hours and time since the vulnerability became known. A mathematical model including both of those vectors only lead to a slight theoretical improvement of the time to remediation [Sha+22] which did not yield enough improvement when it came to real world application. The *Priority Score* approach does not aim to mimic the Exploit Prediction Scoring System (EPSS) while still incorporating some of its research published by the Forum of Incident Response and Security Teams (FiRST).

The following sections will explore how these vectors can be automatically generated from the vulnerability scan results.

4.2.1. Priority Score

A value representing the Probability of Exploitation, the QoD and the CVSS rating are used to calculate a *Priority Score* for each vulnerability and host. This score is made up of the CVSS base score which is modified by two vectors.

1. **QoD** - a vector generated from a percentage value provided by the vulnerability scanner. It indicates the probability of the vulnerability actually existing.

2. **Exploit Probability Indicator** - a value that represents how likely it is to for a vulnerability to be exploited.

Quality of Detection (QoD)

The QoD value is one that already comes with the OpenVAS vulnerability report, but is not used to enhance its vulnerability score. "The quality of detection (QoD) is a value between 0 % and 100 % describing the reliability of the executed vulnerability detection or product detection."[Gre22a] By default the QoD

is set to 70% in OpenVAS. A vulnerability rated with this QoD percentage is defined as "remote_analysis - Remote checks that do some analysis but which are not always fully reliable"[Gre22a]. This ensures as little false positives as possible, while still reporting every service that is potentially prone to exploits. A QoD of 100% is present when a vulnerability has been verified by executing an active exploit. The higher the QoD value has been determined, the higher the probability of the vulnerability actually existing, hence that value is turned into a fraction: $QoD_{new} = \frac{QoD_{old}}{100}$.

Exploit Probability Indicators

Another important vector taken into consideration is the simplicity and therefore likelihood of each vulnerability to be exploited. This takes two factors into account.

Keywords/Tags In order to focus on the most crucial types of vulnerabilities, those that can cause the most harm to the IHE, additional filtering, or in our case modification of the *Priority Score*, had to be performed. To make out these most crucial vulnerabilities, some of the "30 most significant features" [Jac+23] found by FiRST in their report of the third revision of the EPSS, a machine learning based exploit prediction model, where repurposed. The values provided in the report are SHapley Additive exPlanations (SHAP) values[1] derived from their machine learning model on exploit prediction (see [Jac+23, Fig. 7]).

The SHAP values themselves provide no substantial potential for improving the *Priority Score*. Instead the "Tag" texts are used to query the vulnerability descriptions and tags, provided by the OpenVAS report, for their existence. Additionally, 8 additional tags that imply the existence of potentially dangerous vulnerabilities were decided on, which together made up the following list:

- Remote
- Code Execution
- SQLi
- Local
- XSS
- Denial of Service
- Buffer Overflow
- End Of Life
- File Write
- File Deletion

[1] SHAP values are a method used in machine learning to explain the contribution of individual features in predicting model outcomes. They provide a way to quantify the importance of each feature in a unified manner.

- File Modification
- Dangerous Methods
- Dangerous HTTP Methods
- Default Credentials
- Privilege Escalation

The tags inherited from [Jac+23] (highlighted in blue) are used to increase the *Priority Score* by 10 points, since they are proven to be the most influential tags when it comes to exploitation probability, while the additional tags increase it by 5 points. Combinations of the tags can occur, e.g. a vulnerability description with Remote Code Execution would thereby receive 20 additional priority points.

Exploit-DB There are many vulnerability databases in existence such as the National Vulnerability Database (NVD), MITRE's Common Weakness Enumeration (CWE) and Exploit-DB, just to name a few important ones. The NVD and CWE are both extensive collections of known vulnerabilities that comprise of their details such as the impact a successful exploit of a vulnerability can have on the host. What they do not provide are concrete examples or executable exploits [MGS15]. In addition, only about 1-3% of vulnerabilities included in these databases are being exploited in the wild [Alm+17]. This means that a higher or lower probability of exploitation for any of the vulnerabilities listed in those databases cannot be inferred. This is where Exploit-DB deviates from most other vulnerability databases. Along with information from and references to other databases, such as CWE, Exploit-DB also provides exploits for vulnerabilities for which they exist. This includes Proof of Concepts, shellcodes and ready to use exploit code, most of which can easily be leveraged even by inexperienced adversaries through means like metasploit. Even though some of the exploits require more effort to be of use, the risk that comes from a vulnerability that could theoretically be exploited by anyone who wants to, has to be addressed immediately. Vulnerabilities for which at least one entry in the Exploit-DB exists are consequently immediately flagged and put to the top of the prioritized report.

4.2.2. Complexity

Determining the actual time cost of remediation in vulnerability management for each and every vulnerability is not possible with reasonable expenditure, since the

remediation procedure of each vulnerability depends on the context in which it occurs.

With that said, OpenVAS reports include some useful information for that matter. To be specific, the "Solution Type" suggested in those reports can at least give a hint on the amount of work needed for remediation. Five solution types exist in OpenVAS:

- *Vendor Patch* - The vendor released a patch for the vulnerability, usually through an update. The complexity of remediation for this solution type is expected to be very low.
- *Mitigation* - The vulnerability can be remediated by correcting a configuration issue. The complexity of remediation for this solution type is expected to be relatively low.
- *Workaround* - A workaround, that goes beyond configuration, exists. The complexity of remediation for this solution type is expected to be mediocre.
- *No solution exists* - The vendor or a third party has not yet released a solution for the vulnerability, but it is expected that there will be a solution in the future since the service is still receiving updates. The complexity of remediation for this solution type is expected to be high.
- *No fix will be available* - No solution to the vulnerability exists and it is unlikely that there ever will be a fix. This is usually the case if there have not been any updates for the service for a longer period of time or the service has been announced to be discontinued. The complexity of remediation for this solution type is expected to be very high.

To translate these categories into a complexity vector a static value is mapped to each solution type respectively: *Vendor Patch* $\rightarrow$ 1.0; *Mitigation* $\rightarrow$ 2.0; *Workaround* $\rightarrow$ 4.0; *No solution exists* $\rightarrow$ 9.0; *No fix will be available* $\rightarrow$ 10.0.

To substantiate these values the required knowledge, impact on other systems and research time that comes with each remediation technique has to be put into perspective.

Vendor Patch - Updates and security patches can typically be implemented or installed with ease. They generally do not necessitate in-depth knowledge

about the service or the underlying system. Moreover, they should not have any adverse impact on other systems. Hence, a complexity of 1.0 is assigned.

Mitigation - With limited knowledge about the service and clear instructions on where to make modifications, configuring changes does not require substantial effort. Therefore, a complexity of 2.0 is assigned.

Workaround - Workarounds can encompass actions such as disabling specific ports and assessing the potential impact on affected services, as well as installing or deactivating certain services. This necessitates understanding of the system and the services operating on it. The initial complexity is set at 4.0, which may be adjusted subsequently after manual review of the results.

No solution exists - In situations where no viable solution is available, solutions like deactivating the service or adequately isolating it are proposed as workarounds. However, this can lead to compatibility challenges, as other services or hosts may rely on the affected service. Determining the optimal course of action in such cases requires advanced knowledge of the system and a significant amount of time. Therefore, a complexity of 9.0 is assigned.

No fix will be available - When a solution to the vulnerability does not currently exist and will not be available in the future, the ideal approach is to replace the affected service with an alternative that fulfills the same functional requirements. This undertaking demands comprehensive understanding of the system, the service, and the dependencies associated with it. Furthermore, considerable time is expected to be devoted to researching suitable replacement options and executing the migration process. Therefore, a complexity of 10.0 is assigned.

In the following the complexity value is translated 1 to 1 to work hours. The actual time in hours may vary, therefore the assumption is proposed, that, to achieve the actual time, all time values may be scaled by a constant value and thereby any discovery made regarding the time efficiency will keep its validity, since both before and after use the same complexity values.

4.3. Simplification of Results

With priority scores determined for each vulnerability, the order in which existing vulnerabilities should be prioritized is set, but the amount of time it takes to evaluate the report and to address the vulnerabilities remains the same. This is where the simplification approach becomes important. This approach is discussed in this section.

Before data analysis could be performed on the vulnerability scan report, the data was first imported from XML into a SQLite database. Not all fields from the initial XML report, from which a sample result can be seen in A.1, were chosen for the import, since a. some fields are redundant, b. many fields appear in too few results and c. even though the information in a field may be relevant, its content differs greatly across results. The fields chosen for the import are: name [line 2], service (extracted from the name)[line 2], host (as ip)[line 25], hostname [line 27], port [line 29], solution type [line 43], solution text [line 43], severity [line 51], installed version [line 53] and fixed version [line 54].

4.3.1. Vulnerability Scanning

As noted in section 3.2 the scans performed on the institutions networks yielded an expansive amount of information and notably more results than previously estimated. When further breaking down the scan reports into their severity categories high, medium and low, as shown in table 4.1, it becomes clear that some address ranges accumulate many more, especially high severity, vulnerabilities than others.

The reports themselves group vulnerabilities by IP addresses only. Vulnerabilities in the same service on the same host are not grouped together and no hint is given that would suggest that multiple vulnerabilities can be traced back to the same origin. The same applies to vulnerabilities that are caused by network misconfigurations, those are reported for every host affected, even though the vulnerabilities may originate from a router, firewall or another networking device or service.

In addition the reports are not text search friendly, e.g. the keyword ”firewall” is only searchable as ”rewall”. This makes discovering common vulnerabilities

(Scan Nr.) Potential devices	low	medium	high
(1.) 4088	144	207	32
(2.) 3818	171	190	3
(3.) 3320	106	235	122
(4.) 3320	311	440	136
(5.) 4070	218	115	31

Table 4.1: Vulnerability Scan Result Severities, Grouped by Scan

within each host or across multiple hosts tedious and non feasible.

4.3.2. Data Categorization

In order to allow aggregation of the data provided by a vulnerability scan report it is important to categorize the fields of the results within those reports. To identify fields that are worth categorizing, a closer look into the report's content and structure is vital. The following categories materialized from different grouping and filtering of the results inside the database.

Updates

When inspecting some hosts in the PDF format report, one main observation was made. There were multiple vulnerabilities caused by the same service which are fixed in different versions. This is of course not unexpected, a quick lookup for MySQL version 6 in searchsploit resulted in five vulnerabilities, SMB3.1.1 in two vulnerabilities, PHP 7.0 in nine vulnerabilities and Samba 2.2.8 in six vulnerabilities. Versions of services for which at least one known vulnerability is known get tested more thorough than others and thereby more vulnerabilities for them are detected.

In order to identify these updates the SQL query, listing 1, was constructed. The resulting updates are listed with the oldest detected version of the running service as well as the version which should remediate all detected vulnerabilities of that service. Filtering by solution texts which start with "update" and solution

```sql
SELECT ip, service, port, MIN(installed_version), MAX(fixed_version), COUNT(*)
FROM results
WHERE solution_text LIKE "update %"
    AND solution_type = "VendorFix"
GROUP BY ip, service, port
```

Listing 1: SQL Query: Discovering Updates

IP (anonymized)	Service	Port	Oldest Version	Fixed Version	Count
127.0.0.6	Oracle	3306/tcp	5.7.17	5.7.41	43
127.0.0.46	Oracle	3306/tcp	5.6.34	5.7.41	29
127.0.0.86	Apache	80/tcp	2.4.17	2.4.56	25
127.0.0.34	ownCloud	443/tcp	8.1.9	10.8	8
127.0.0.57	Atlassian	8090/tcp	5.2.3	6.5.2	4

Table 4.2: Excerpt of Identified Updates

types which are reported as "VendorFix" makes sure only updates are included in the results. Counting the number of rows for each result of the query showed that indeed many vulnerabilities on the same hosts can be remediated by single updates as can be seen in table4.2, with the associated, anonymized IPs [2].

Updates discovered by this technique will be included in the final report as "Mandatory Updates".

Minor Updates Table 4.2 shows only a small portion of identified updates, but one fact allowed separating them further: the difference between the oldest and the fixed version. While some updates require a major version change, i.e. versions that might introduce incompatible API changes, others require a minor version change, i.e. versions that add backward compatible functionality [Prend].

Updates that lie within the minor version changes were identified by adding

```sql
HAVING substr(installed_version, 0, instr(installed_version, ".")) =
```

[2]The IPs are randomized and not hashed as even low end CPUs take merely 4 seconds to hash $4 \cdot 10^9$ IP addresses[Lat22]. This would allow computation of the real addresses.

IP (anonymized)	Service	Port	Severity	Count
127.0.0.4	PHP	80/tcp	10.0	44
127.0.0.67	OpenSSL	80/tcp	10.0	16
127.0.0.77	SMB	445/tcp	10.0	1
127.0.0.98	phpBB	80/tcp	9.9	1
127.0.0.13	Lighttpd	80/tcp	9.8	1

Table 4.3: Excerpt of High Severity Vulnerabilities

```
substr(fixed_version, 0, instr(fixed_version, "."))
```

to the end of the query. Which checks for equality of the major version between the installed and fixed versions.

These updates will be included in the final report as "Minor Updates".

High Severity

Another important category of vulnerabilities is that of high severity vulnerabilities. These vulnerabilities are of high concern to the IHE as they pose an imminent risk to its network's and services' confidentiality, integrity and availability.

```
1 SELECT ip, vulnerable_service, port, severity, COUNT(*) as cnt
2 FROM results
3 WHERE severity >= 7.0
4 GROUP BY ip, vulnerable_service, port
```

Listing 2: SQL Query: High Severity Vulnerabilities

Vulnerabilities which lie within the high severity range can be determined with the query in listing 2. While the results of the query show some services for which multiple high severity vulnerabilities were found, most of them were detected with a single high severity vulnerability. Therefore almost no reduction in terms of remediation time can be made here, but they are important nevertheless and will be included in the final report as "High Severity Vulnerabilities".

It is important to not that the query presented here does not take advantage of the previously discussed priority score, this score was used later in chapter 5.2.2 as the score is just determined before that.

Frequently Occurring

One prominent initial observation in the report was the appearance of a couple of vulnerabilities on multiple hosts. This was also discovered by [CDN20]. The query 3 was constructed to confirm the observation on a larger scale.

```
SELECT name, solution_text, COUNT(*) as cnt, MAX(severity)
FROM results
GROUP BY name
ORDER BY cnt DESC
```

Listing 3: SQL Query: Frequently Occurring Vulnerabilities

Executing the query on the given data confirms the initial observation, as table 4.4 shows. While the CVSS score of the vulnerabilities ranks most of the widely spread vulnerabilities as low to medium severity, ”chaining” of such vulnerabilities can be leveraged increase their impact way beyond their initial CVSS rating [CIS21]. Inspecting the solution texts of these vulnerabilities reveals that many of them can be remediated by changes to the firewall or by disabling specific algorithms.

Adding

```
WHERE solution_text LIKE "%firewall%" OR solution_text LIKE "disable%"
```

to the query reveals such cases. The following solution text provides an example for this:

```
  Various mitigations are possible:
- Disable the support for ICMP timestamp on the remote host completely
- Protect the remote host by a firewall, and block ICMP packets passing
through the firewall in either direction (either completely or only for
untrusted networks)
```

Since larger institutions usually utilize enterprise grade firewalls which allow to block specific packages, in this case Internet Control Message Protocol (ICMP)

Vulnerability Name	Count	Max Severity
ICMP Timestamp Reply Information Disclosure	545	2.1
DCE/RPC and MSRPC Services Enumeration Reporting	335	5.0
TCP timestamps	290	2.6
SSL/TLS: Deprecated TLSv1.0 and TLSv1.1 Protocol Detection	261	4.3
ICMP Netmask Reply Information Disclosure	85	2.1

Table 4.4: Excerpt of Frequently Occurring Vulnerabilities

packages. This means the fix to those 545 vulnerabilities could be distributed with a single configuration change.

The IHE in question split the network in multiple DMZs, which would require additional grouping of the data by DMZ, subnet or, as elaborated in sections 5.2.1 and 5.2.2, faculty.

4.4. Expectancy Towards the Report

The goal of the simplified report is to provide an overview of vulnerabilities and their remediation techniques, that is not overwhelming to the responsible person. It *has to* include vulnerabilities that can be remediated by simply installing an update, vulnerabilities that have a high priority, vulnerabilities that occur relatively often and those that need attention but which remediation is expected to be very complex. The vulnerabilities listed within the prioritized report *have to* cover the majority of vulnerabilities from the original report while minimizing the cost of remediation. The report *should* be divided by area of responsibility, e.g. by faculties, system administration, etc. The report *has to* make clear in which order the vulnerabilities should be remediated, provide details on the vulnerabilities, such as its' possible effect on the system, which IP addresses are affected and how to remediate the vulnerability, and highlight vulnerabilities that need immediate attention due to either the type of vulnerability (e.g. Remote Code Execution (RCE), Default

Credentials, etc.) or the existence of automated exploits for that vulnerability.

Leading the report *has to* be a list of "Minor Updates" which *should* include all vulnerabilities which are flagged by the vulnerability scanner as "Vendor Patch" and which only require a minor version change as explained in section 4.3.2. Remediations listed in this section can be delegated to personnel with little to no knowledge about the host.

The list of "Major Updates" *should* include all vulnerabilities that are flagged by the vulnerability scanner as "Vendor Patch" or include a version to update to, if multiple vulnerabilities exist for the same service on the same host, the version that fixes all vulnerabilities for such service *has to* be provided to reduce the time it takes to find the right version to update to.

Following the "Major Updates" section, the "High Priority" section *has to* include all vulnerabilities that have a *Priority Score* (see section 4.2.1) of at least 7.0 for which no updates exist. This *has to* include all confirmed high, CVSS score of $7.0 - 8.9$ and a QoD of $\sim 100\%$, and all critical vulnerabilities, CVSS score of $9.0 - 10.0$ and a QoD of $70 - 100\%$. Additionally it *should* include easily exploitable vulnerabilities. (see section 4.2.1) Vulnerabilities with a high *complexity* (see section 4.2.2) *should* not be included in this section.

The fourth section of "Frequently Occurring Vulnerabilities" *should* cover all vulnerabilities that exceed a threshold of hosts and services affected within each area of responsibility. It can be expected, that even though the first remediation of such vulnerabilities may take considerable amount of time, the following remediations of the other hosts or services will be much lower, or the remediation could be a one fix for all scenario where the remediation can be performed for the entire system or network at once.

Finally the section of "High Effort Remediations" *should* include those vulnerabilities that are expected to require a lot of resources for remediation.

Vulnerabilities covered in a previous section *should* not be listed in the following sections, since the report is expected to be worked through from top to bottom, which *should* provide optimal cost-benefit efficiency and a fast way to remediate the most critical issues.

In addition to all categorized vulnerabilities, which *should* have an increased efficiency (see section 4.1) compared to the initial report, those vulnerabilities that do not fit into either category *should* be included at the end of each report to avoid providing a false sense of security.

A search or filter functionality *can* be added to simplify the remediation process further.

5

Methodology

The purpose of this chapter is to present the methodology employed in this thesis to address the research objective of vulnerability prioritization and mitigation within the context of an institution of higher education.

The full procedure that leads from a list of subnets to the final simplified report is shown in figure 5.1. This chapter outlines the step-by-step approach followed to effectively scan and analyze vulnerabilities using the OpenVAS scanner, and subsequently evaluate the results through data analysis and simplification techniques.

5.1. Prerequisites

The entire script is tested under kali linux since the OpenVAS scans were run there as well.

In order to perform any steps of the result prioritization, the IHE provided information about and access to its networks. All result prioritization is performed on scan results exported from OpenVAS version v22.6.1.[1]

First of all, the IHE specified the vulnerability scan target. This could have been provided in the form of a list of IP addresses or subnets and a list of ports

[1] https://github.com/greenbone/openvas-scanner/releases/tag/v22.6.1

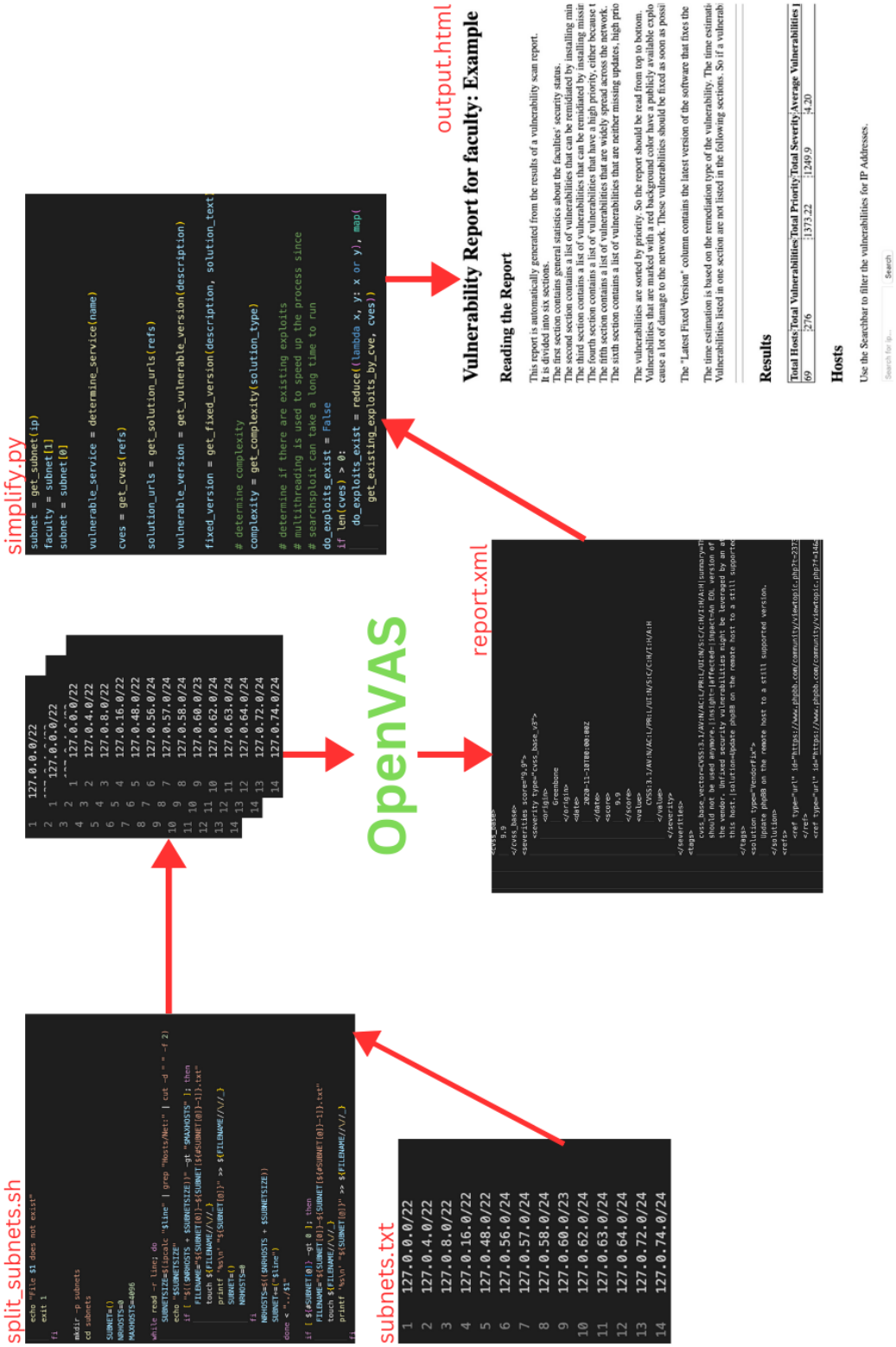


Figure 5.1: Simplification Pipeline

to scan. A list of subnets was the starting point in this case. Since no list of ports was provided and a thorough scan was requested, the top 5000 Internet Assigned Numbers Authority (IANA) assigned Transmission Control Protocol (TCP) ports available in OpenVAS were used.

Furthermore, during the entire process, a files that map *Subnets* to *Faculties* (see A.2) and *Faculties* to *Contact E-mails* (see A.4) were utilized throughout different parts of the process.

Subnet Partitioning and Scan Configuration Since "[the] maximum configurable number of IP addresses is 4096"[Gre22b] in OpenVAS, networks with a subnet-mask larger than /20 have to be split up into sets of /20 networks. For the HTWG with a class B network (/16 subnet mask), this would mean that 16 scans have to be performed. Fortunately only a fraction of all available IP addresses are actually in use, 18,944 IPs in 36 subnets ranging from /24 to /22 to be specific. This means the entire IHE network can be scanned in 5 separate vulnerability scans. To assure maximum realistic coverage all scans were performed from within the IHE's DMZs.

In order to simplify the subnet splitting process a bash script was developed that automates the process of determining the IP address count on the provided list of subnets and splits them up accordingly. (see B.1) The script takes a file with a list of subnets in the CIDR notation format as its only argument, determines the number of hosts for each subnet, splits them up into chunks of a maximum of 4096 hosts and writes them to multiple files accordingly. Note that `ipcalc` has to be installed on the system for the script to work as expected.

The "Full and Fast" OpenVAS scan configuration was chosen in order to cover the maximum amount of potential vulnerabilities in a reasonable time-frame. All scans were performed unauthenticated as it provides a real-world portrayal of existing vulnerabilities from an adversary perspective [Hol+11]. There are several categories within the scan configuration, most of them were left unchanged. The first setting that needed to be changed was in the "Nmap (NASL wrapper)" section. After testing several timing policies, the "Aggressive" behaviour was chosen since it performs the fastest without being interrupted by a firewall, this may vary between IHEs. Additionally, since institutions of higher education have to comply with the "IT Grundschutz", the "Compliance Tests" are also activated. For the "Brute force attacks" category, all but the default credential attacks have been

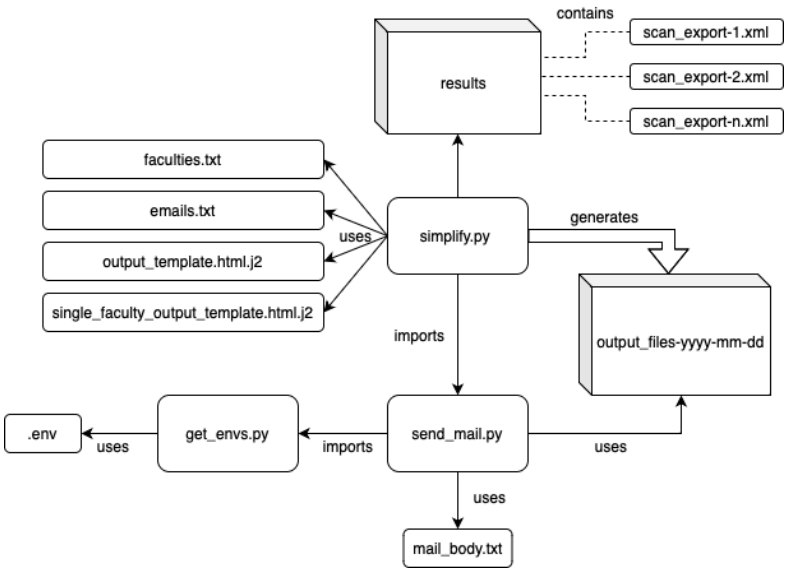


Figure 5.2: Program Structure

disabled to prevent password resets or accidental denial of services.

It has to be noted that the OpenVAS scan configuration "Full and Fast" caused a Denial of Service (DoS) on a small number of hosts even though the description of the configuration explicitly states that "[only] VTs that will not damage the target system are used"[Gre22b]. (VTs stands for Vulnerability Tests in this context)

5.2. Implementation of the Simplification Framework

For the implementation Python3 (version 3.11) was chosen as the primary General Purpose Language (GPL) since it provides native XML-Element-Tree and SQLite3 support. To ensure longevity and out-of-the-box usage of the script, only packages included with Python3 were used. The Python scripts that were developed perform three main steps to create the final simplified report. First, the XML files of all scans exported from the OpenVAS scanner undergo a pre-processing pipeline which also inserts its data into a SQL database. Second, the data gets analyzed and prioritized with SQLite within the database. And in the final step a comprehensive, simplified and prioritized report is generated from the data in the database and sent out to the responsible personnel.

Figure 5.2 depicts the project structure of the python script B.5. The *results*

folder contains all exported scan results in XML format. The files *faculties.txt*, *emails.txt*, *output_template.html.j2* and *single_faculty_output_template.html.j2* (see A.3, A.4, A.1 respectively) have to be present in the working directory of the script. *send_mail.py*, *get_envs.py*, *mail_body.txt* and *.env* also need to be present as they allow for the distribution of the results through e-mail. Since the implementation of the e-mail sending scripts may vary without affecting the results of this thesis, only a simple implementation is provided (see B.6, B.7, A.5).

The following line numbers all refer to B.5.

5.2.1. Vulnerability Pre-Processing

The script is initially provided with access to the data contained in the XML files [lines 293-319]. Most information required is accessible within the XML element-tree as text attribute [lines 348-372]. The values "vulnerable_version", "fixed_version", "priority", "complexity", "subnet", "faculty" and "exploit_exists" require additional processing before being used.

The vulnerable and fixed version can be found and captured from the vulnerability description with regular expression [lines 146-161, 166-204]. An exception exists for the fixed version if the version stated is "eol version", in this case "99.99.99" is assigned as its value as an End of Life (EOL) indicator to remain comparable in SQLite. The value is later replaced as described in section 5.2.3. Determining the subnet the vulnerable host belongs to is important so that it can then be associated with its corresponding faculty [lines 209-215, 384-386]. A file that maps subnets to faculties is required in the form depicted in A.3. Following the complexity mapping developed in section 4.2.2, each vulnerability is assigned its complexity accordingly [lines 59-63, 248-260]. The existence of an exploit for a vulnerability is evaluated by consulting searchsploit for each CVE belonging to the vulnerability [lines 269-288]. Each vulnerability is then evaluated based on the factors described in section 4.2.1 in order to assign a *priority score* [lines 410-422] prior to being inserted into a database [lines 425-461] for later processing.

The objective of the pre-processing step is to attain all data required to conform with the database schema seen in figure 5.3. Since the database is needed for analytic purposes only, a *star schema* was chosen. It allows for easy read access with only a small JOIN overhead [Ecknd]. Apart from marking a *result* as

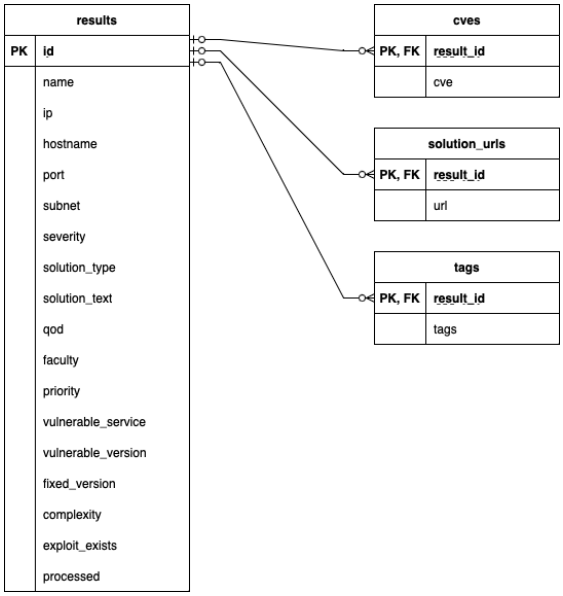


Figure 5.3: Initial Database Schema

”processed”, the initial database is never altered.

5.2.2. Vulnerability Prioritization and Analysis

Central to the prioritization script are five comprehensive SQL queries. These queries each generate one table for each part of the final report which are ”minor_updates”, ”major_updates”, ”high_priority_vulnerabilities”, ”wide_spread_-vulnerabilities” and ”high_effort_remediations”. All five queries follow a similar pattern of collecting all CVE entries and solution URLs for a vulnerability name and IP or faculty.

Since all five queries share the same structure, their behaviour will be explained with the example of the ”major_updates” table query, see B.5 [lines 580-625].

The query begins by usi ng two Common Table Expressions (CTEs): ”tmp_cves” and ”tmp_-solution_urls.” These CTEs are temporary result sets that will be used in the subsequent query.

The ”tmp_cves” CTE selects the ”result_id” and a concatenated string of distinct CVEs associated with each result. It retrieves this information by joining the ”results” table with the ”cves” table on the ”id” column, and then grouping the results by the IP address, service and port. The ”tmp_solution_urls” CTE follows a similar pattern but retrieves the ”result_id” and a concatenated string of distinct

solution URLs associated with each result instead.

The main query then selects various columns from the "results" table and performs several aggregate functions on other columns. These include calculating the maximum, sum, and average of the "priority" column. It also counts the number of occurrences of the vulnerabilities. Additionally, it retrieves a concatenated string of distinct solution texts and selects the minimum vulnerable version, maximum fixed version, i.e. the version to update the service to, the list of CVEs from the "tmp_cves" CTE, the list of solutions from the "tmp_solution_urls" CTE, maximum of the column "exploit_exists", i.e. at least one exploit exists for the service in question, and the maximum complexity.

In order to only include vendor fixes which are effectively updates or upgrades, the results get filtered by their "solution_type" to be equal to "VendorFix" and the "solution_text" to contain "update %" or "upgrade %" (ignoring case).

The query then groups the results by the "ip" and "name" columns. Finally, the results are ordered by the "faculty" column, with the "exploit_exists" column in descending order and the "max_priority" column in descending order as well.

Once the table is created, the "results" tables "processed" column entries of the results included in the newly created table are updated to be ignored by the following queries.

Differences between Queries To retrieve the widely spread vulnerabilities the results are not grouped by IP addresses, but by faculty names instead. Thus the corresponding ip and host_name columns have to be concatenated, otherwise they would be lost.

For the high priority vulnerabilities only the where clause changes to only include vulnerabilities over a certain threshold but exclude those above a high cost threshold of 8.0. Those vulnerabilities that cross this high cost threshold are then included in the high effort remediations.

Time Complexity A queries time complexity is a crucial factor when evaluating the efficiency of a framework that is focused on saving time. Therefore the SQL query "major_updates", as described before, underwent a small set of tests. For these tests the "results" table was filled with randomized entries, starting with 1,000 total entries up to 1,000,000 entries, and the time for execution of the "major_updates" query was measured. The results, as seen in table 5.1, suggest a time

results table entries	1,000	10,000	100,000	1,000,000
time for execution	34ms	66ms	264ms	2432ms

Table 5.1: Execution Time for major_updates Query

complexity of $\mathcal{O}(n)$. Another test with 10,000,000 entries was performed which took 36036ms. This spike was traced back to the computer running out of main memory which caused slower computation.

Formalizing the time complexity was done by analyzing the query plan. The output of adding

```
EXPLAIN QUERY PLAN
```

to the beginning of the query is seen in listing 4. Crucial to the time complexity are the lines 2, 7 and 11 as `SCAN table` iterates through every table entry. This results in $n + c_1 \cdot n + c_2 \cdot n$ table lookups or a time complexity of $\mathcal{O}(n)$ as c_1 and c_2 are small constant values as the tables "cves" and "solution_urls" each contain an average of one to three entries per result entry. `B-TREE`, or binary tree, searches take logarithmic time, time complexities of $\mathcal{O}(log_2 n)$, therefore they do not weigh into the time complexity as $n \gg log_2 n$ holds for large n.

Threshold Evaluation

The script enables the benchmarking of two thresholds that are crucial in the data analysis queries. The first threshold, referred to as the *WIDE_SPREAD_THRESHOLD*, plays a vital role in categorizing a vulnerability as *frequently occurring* based on the number of occurrences within a faculty. On the other hand, the second threshold, referred to as the *HIGH_PRIORITY_THRESHOLD*, sets a threshold for the *Priority Score* assigned to a vulnerability to be considered as *high priority*. Evaluating the performance of these thresholds involves conducting tests where one threshold remains at its default value while the other is assigned different values within a defined range. The prioritization and analysis of results are performed for each value in the range, and the performance indicators are saved for comparison.

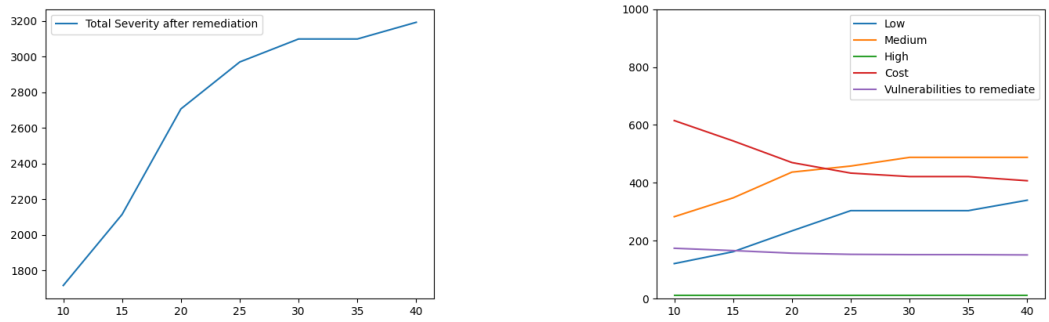
In the evaluation of the Wide Spread Threshold (WST), a range of values from 10 to 40, with a step size of 5, has been utilized. Figure 5.4b illustrates the impact of modifying the WST on the coverage of medium severity vulnerabilities. It is evident that deviating from a WST of 10 leads to a significant decrease in

```
MATERIALIZE tmp_cves
SCAN c
SEARCH r USING INDEX result_ids (id=?)
USE TEMP B-TREE FOR GROUP BY
USE TEMP B-TREE FOR group_concat(DISTINCT)
MATERIALIZE tmp_solution_urls
SCAN s
SEARCH r USING INDEX result_ids (id=?)
USE TEMP B-TREE FOR GROUP BY
USE TEMP B-TREE FOR group_concat(DISTINCT)
SCAN r
SEARCH c USING AUTOMATIC COVERING INDEX (result_id=?)
SEARCH s USING AUTOMATIC COVERING INDEX (result_id=?)
USE TEMP B-TREE FOR GROUP BY
USE TEMP B-TREE FOR group_concat(DISTINCT)
USE TEMP B-TREE FOR ORDER BY
```

Listing 4: SQL Query Result: Explain Query Plan

coverage. Specifically, at a threshold of 10, there are 415 vulnerabilities remaining, whereas at a threshold of 40, the number increases to 839. However, increasing the threshold only results in a modest improvement in time effort, with 615 hours required at a WST of 10 and 407 hours at a WST of 40. Moreover, the reduction in the total number of vulnerabilities to be remediated is marginal, comparing 174 vulnerabilities at a WST of 10 to 151 at a WST of 40. Interestingly, the total severity after remediation increases by 86%, from 1716.0 to 3191.8 (see figure 5.4a). This great difference in the total severity with only relatively little difference in the number of vulnerabilities to remediate can be traced back to many very widely spread vulnerabilities which were summarized into a hand full of fixes due to their similarity. Thereby the number of remaining vulnerabilities decreases significantly with only little additional vulnerabilities to remediate, while the time needed to remediate all vulnerabilities is still accounted for.

By evaluating the trade-offs between vulnerability coverage, time effort, and total severity after remediation, it becomes apparent that a lower WST value of 10 strikes an ideal balance. Although increasing the WST might provide a slight improvement in time efficiency, it leads to a higher number of vulnerabilities that need to be addressed and potentially increases the severity of the remaining vulnerabilities. Therefore, a WST of 10 ensures that vulnerabilities are adequately



(a) Total Severity after remediation

(b) Cost and Remaining Vulnerabilities

Figure 5.4: Wide Spread Threshold Statistics

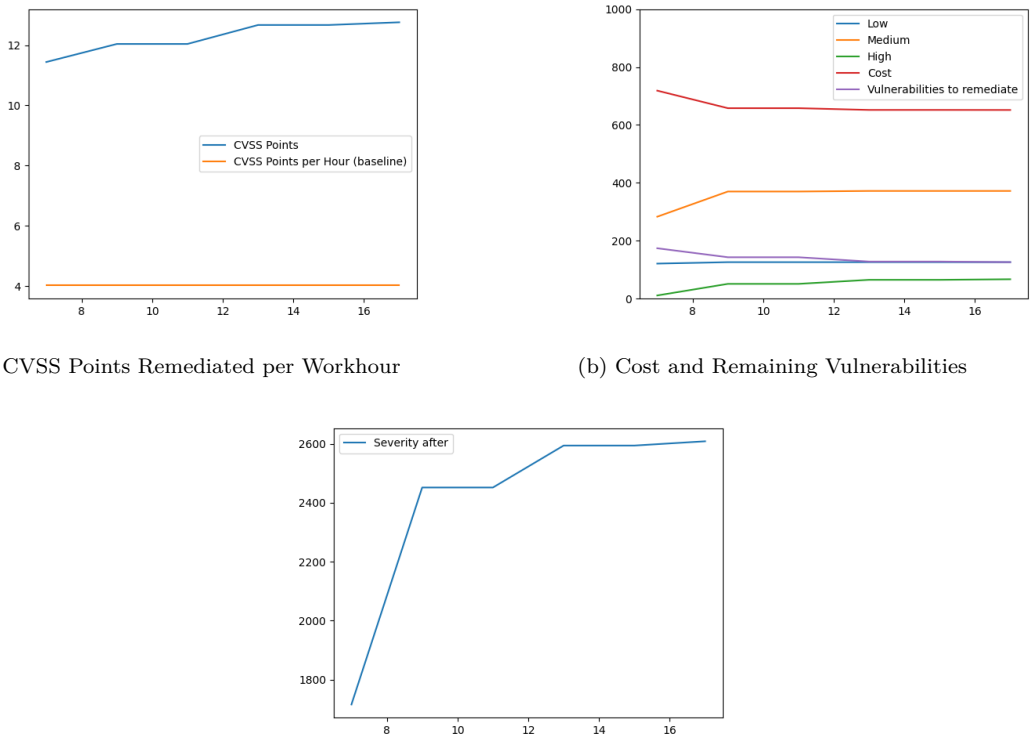
categorized as "wide spread" without unnecessarily burdening the remediation process or compromising the overall severity reduction achieved. This value ensures a practical and effective prioritization strategy for vulnerability management within the studied context. Consequently, a default WST value of 10 has been assigned based on these findings.

In the evaluation of the High Priority Threshold (HPT), a range of values from 7 to 19, with a step size of 2, has been utilized. Although intermediate steps and different ranges were experimented with, they did not yield any further insights.

When analyzing figure 5.5b, a significant issue becomes apparent. Increasing the HPT only slightly from 7 to 9 results in a small percentage decrease in the cost of remediation, specifically 8.4%. However, the number of remaining medium and high priority vulnerabilities increases significantly by 30.7% and 463.6%, respectively. This behavior is also reflected in the remaining severity, which rises by more than 700 points or 42.9% (see figure 5.5c).

In this context it is crucial to ensure that lowering the HPT does not have a negative impact on the cost-benefit relationship. To address this concern, an examination of the CVSS points remediated per work-hour was conducted.

Figure 5.5a reveals an interesting trend: including more vulnerabilities by lowering the HPT actually increases the effectiveness of each work-hour in terms of CVSS points remediated. This finding contradicts the initial expectation and suggests that the benefits of including additional vulnerabilities outweigh the potential, in this case marginal, increase in workload. The trade-off is justifiable as the slight increase in workload is compensated by the improved efficiency in terms of CVSS points remediated per work-hour. As a result, the decision was made to



(a) CVSS Points Remediated per Workhour

(b) Cost and Remaining Vulnerabilities

(c) Severity after Remediation

Figure 5.5: High Priority Threshold Statistics

set the HPT at 7.

5.2.3. Report Creation

After the vulnerabilities from the report have been analyzed and prioritized accordingly, the five tables have to be made available to the system administrators of each faculty in an easily readable format.

To accomplish this, two different HTML Jinja[2] templates were designed in which the data is then added. The first template, see A.1, is used for each individual faculty. It includes an introduction on how to read the report, a search function to filter the report by an IP address, an overview of the faculty's vulnerability statistics and five tables, for the five categories of vulnerabilities explained in the previous section, which include all necessary information that the system ad-

[2]https://jinja.palletsprojects.com/en/3.1.x/

ministrators need for remediation. In addition to the five tables another optional table exists which includes all uncategorized vulnerabilities. This table is only shown when there exist no categorized vulnerabilities for the faculty. Otherwise a hint is shown which indicates the number of vulnerabilities left which can be "unlocked" by remediating all other vulnerabilities first. This approach of *gamification*[3] was chosen to motivate the person working on the report by providing a milestone that is achievable. The approach of gamification has been discovered to be of great value when it comes to enjoyment and productivity in a work environment [Ger+20].

The second template, see A.6, is a version of the first one which includes all faculties in one document to be sent to the IT security officer of the institution.

The process of report creation is split into three main steps. In the first step, the data from all essential tables is extracted from the database and saved into an object [lines 1067-1098], since the data is already sorted and grouped as required, no further SQL data manipulation has to be done. The second step consists of rearranging the retrieved data and setting it up to be used as input to the templates [lines 1103-1193]. This includes concatenating IP addresses, host names and CVEs and also to replace previously as version 99.99.99 saved versions with an appropriate explanation. Afterwards statistics for each IP address are generated including their maximum priority score and count of remediations split up into each of the five categories [lines 1196-1216]. This data is then fed to the HTML templates and rendered as standalone HTML files in the third step [lines 1218-1231, 1239-1248].

After each report is created, it is sent to the system administrator responsible for the faculty via e-mail [lines 1236-1238, 1252-1254].

[3]The Oxford Dictionary defines gamification as "the application of typical elements of game playing (e.g. point scoring, competition with others, rules of play) to other areas of activity, typically as an online marketing technique to encourage engagement with a product or service".

6

Results

As described in section 5.2 the process of simplification focuses on choosing the right vulnerabilities to remediate and group vulnerabilities that can be remediated in one go. This leads to a great reduction in size and effort for remediation. In this chapter the results of the simplification process are discussed.

6.1. Usefulness

In order to compare the characteristics of usefulness, described in section 4.1, as well as other statistics that contribute to better understanding the results, the baseline of those values was calculated.

The effectiveness of the pre-simplification report is as expected 1.0, since the report schedules all vulnerabilities for remediation, which is, as deducted in chapter

	effectiveness	**efficiency**
baseline	1.0	4.02
after simplification	0.87	11.44

Table 6.1: Baseline effectiveness and efficiency in CVSS points per hour

	high	medium	low
baseline	324	1187	950
addressed in simplified report	313	904	829
remaining	11	283	121

Table 6.2: Vulnerabilities by severity, before and after

3, way above an IHE's capabilities. The efficiency calculated from the initial report is 4.02 CVSS points per hour, which corresponds to an average remediation of one medium severity vulnerability per hour. After performing the simplification script on the same data the effectiveness and efficiency change as seen in table 6.1. While the effectiveness drops by 0.12, the efficiency almost triples to 11.44 CVSS points per hour.

The decrease in effectiveness is the result of the result prioritization described in sections 4.2.1 and 5.2.2, as some vulnerabilities do not fit in either criteria of the five sections of minor updates, mandatory updates, high priority vulnerabilities, wide spread vulnerabilities and high effort remediations. Table 6.2 depicts these uncategorized vulnerabilities. Within those vulnerabilities the five most common ones are as follows (ordered by number of occurrences):

1. ICMP Timestamp Reply Information Disclosure
2. ICMP Netmask Reply Information Disclosure
3. SSL/TLS: Report Weak Cipher Suites
4. SSL/TLS: Deprecated TLSv1.0 and TLSv1.1 Protocol Detection
5. DCE/RPC and MSRPC Services Enumeration Reporting

Even though those vulnerabilities occur multiple times, they are spread across multiple faculties and are thereby not included in the wide spread category.

The large increase in efficiency can be traced back to multiple factors. First of all, in the final report just 174 vulnerabilities need to be remediated to address 83.14%, or 2,046 out of 2,461, vulnerabilities. This includes wide spread vulnerabilities which might require more than just one remediation for a large group of vulnerabilities. Therefore the time for their remediation was adjusted, for each

reoccurence of a wide spread vulnerability 20% of its initial remediation cost was added to its total remediation time. Even with this adjustment the 83.14% of vulnerabilities are addressed by the report with a predicted time effort of just 15,12%, or 718.42 of an initial 4,752.0 hours, of the initial total time to remediation, with the remaining vulnerabilities estimated at a 901 hour remediation time. Time savings through grouping of vulnerabilities are maximized in the wide spread, minor- and missing updates sections which address 36.26, 7.89 and 5.23 vulnerabilities per remediation step respectively. Meanwhile the high priority vulnerabilities and high effort remediations address 3.13 and 1.67 vulnerabilities.

6.2. Report Length

The simplified report generated by the prioritization script comes in an HTML format and not as PDF and thereby cannot be directly compared to the initial report regarding its length. Nonetheless when exporting the report with a browser's print function the resulting PDF file is about 110 pages long.[1] This results in an average 5.2 pages per faculty, with 20 faculties included in this scan. An excerpt of a faculty report can be seen in A.6. Since the vulnerability report is now present in HTML format text search is possible throughout the report, even though this should not be required as a built in search function serves to filter the scan results by IP address. This allows system administrators to address all vulnerabilities on a single host in one go without the distraction of unnecessary information.

Evaluation of the report, i.e. determining urgency as well as association between responsibilities and vulnerabilities, is already performed, reducing the immense time cost discussed in chapter 3 to a couple of minutes, thereby freeing up resources for the actual remediation procedure.

[1] The length may vary depending on screen-size and browser.

7

Conclusion and Future Work

Conclusion The implemented tool is capable of reducing the time required for evaluating a vulnerability scan report significantly. This frees up resources which can be used for remediation, by automatically prioritizing the most imminent remediations and simplifying the remediation process through merging of similar or connected vulnerabilities and categorizing them by the type of remediation. The tool is self-contained and should not require any updates in the near future when used in combination with the appropriate vulnerability scanner version.

The final simplified report includes a fraction of the remediation steps from the initial report but still addresses the majority of vulnerabilities.

Future Work Subsequent research should focus on further improving the time estimation for vulnerability remediation. A natural language processing model is suitable for this purpose, both for identifying further distinctions within frequently occurring vulnerabilities and to derive, not an estimation, but the actual time needed for a vulnerability's remediation from its provided information.

Similar approaches should be considered to locate vulnerabilities and services which require external expertise or which are generally considered to be ideally hosted by experts as the cost of providing sustained on-premise security is not commensurate.

The current implementation could be enhanced with the addition of a (web-)interface to avoid distributing emails and thereby allowing for a centralized vul-

nerability management which would make monitoring the remediation progress much easier for the information security officer.

As of now only sporadic remediation attempts were performed based on this thesis' results, extensive vulnerability management in real world scenarios should be performed to evaluate the accuracy of the calculated work-hour performance increases.

In addition to simplifying scan results, an approach is proposed which provides guidance for each vulnerability on how to prevent their occurrence henceforth and thereby ensures long-lasting protection.

A

Data

A.1. Single Result


```xml
<result id="XXXX-XXXX-XXXX-XXXX-XXXX">
    <name>Oracle MySQL Server &lt;= X.XX.XX / X.X &lt;= X.XX.XX</name>
    <owner>
        <name>admin</name>
    </owner>
    <modification_time>2023-04-25T10:45:12Z</modification_time>
    <comment></comment>
    <creation_time>2023-04-25T10:45:12Z</creation_time>
    <detection>
        <result id="XXXX-XXXX-XXXX-XXXX-XXXX">
            <details>
                <detail>
                    <name>product</name>
                    <value>cpe:/a:vendor:service:X.XX.XX-log</value>
                </detail>
                <detail>
                    <name>location</name>
                    <value>3306/tcp</value>
                </detail>
                <detail><name>source_oid</name><value>1.1.1.1.1.1.1.1</value></detail>
                <detail><name>source_name</name><value>MySQL</value></detail>
            </details>
```

```
23          </result>
24      </detection>
25      <host>127.0.0.1
26          <asset asset_id="XXXX-XXXX-XXXX-XXXX-XXXX"></asset>
27          <hostname>example.com</hostname>
28      </host>
29      <port>3306/tcp</port>
30      <nvt oid="1.1.1.1.1.1.1.1.1">
31          <type>nvt</type>
32          <name>Oracle MySQL Server &lt;= X.XX.XX / X.X &lt;= X.XX.XX</name>
33          <family>Databases</family><cvss_base>9.8</cvss_base>
34          <severities score="9.8">
35              <severity type="cvss_base_v3">
36                  <origin>NVD</origin>
37                  <date>2021-08-31T16:37:00Z</date>
38                  <score>9.8</score>
39                  <value>CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H</value>
40              </severity>
41          </severities>
42          <tags>Lorem Ipsum</tags>
43          <solution type="VendorFix">Update to version X.XX.XX, X.XX.XX or later.</solution>
44          <refs>
45              <ref type="cve" id="CVE-2021-3711"></ref>
46              <ref type="url" id="https://www.example.com/fix"></ref>
47          </refs>
48      </nvt>
49      <scan_nvt_version>2021-10-23T08:58:44Z</scan_nvt_version>
50      <threat>High</threat>
51      <severity>9.8</severity>
52      <qod><value>80</value><type></type></qod>
53      <description>Installed version: X.XX.XX
54                  Fixed version:     X.XX.XX
55                  Installation
56                  path / port:       3306/tcp
57      </description>
58      <original_threat>High</original_threat>
59      <original_severity>9.8</original_severity>
60  </result>
```

A.2. subnets.txt

```
127.0.0.1/24
127.0.0.2/24
127.0.0.4/22
...
```

A.3. faculties.txt

```
127.0.0.1/24|Faculty_Name_1
127.0.0.2/24|Faculty_Name_2
127.0.0.4/22|Faculty_Name_3
...
```

A.4. emails.txt

```
Faculty_Name_1|faculty1@example.com
Faculty_Name_2|faculty2@example.com
Faculty_Name_3|faculty3@example.com
...
```

A.5. .env

```
EMAIL_ADDR=it-sec@example.com
PASSWORD=**password**
EMAIL_HOST=smtp.example.com
EMAIL_PORT=587
```

A.6. Sample Report

Vulnerability Report for faculty: Example

Reading the Report

This report is automatically generated from the results of a vulnerability scan report.
It is divided into six sections.
The first section contains general statistics about the faculties' security status.
The second section contains a list of vulnerabilities that can be remediated by installing minor updates.
The third section contains a list of vulnerabilities that can be remidiated by installing missing updates.
The fourth section contains a list of vulnerabilities that have a high priority, either because they are exploitable or because they have a high priority score.
The fifth section contains a list of vulnerabilities that are widely spread across the network. Usually these vulnerabilities are caused by a misconfiguration of the network or a widely used service.
The sixth section contains a list of vulnerabilities that are neither missing updates, high priority nor widely spread, but are expected to require a lot of time to fix.

The vulnerabilities are sorted by priority. So the report should be read from top to bottom.
Vulnerabilities that are marked with a red background color have a publicly available exploit or a priority score of 10 or higher. A maximum priority of above 10 is usually caused by vulnerabilities like Remote Code Execution, Default Credentials or other vulnerabilities that, when exploited, can cause a lot of damage to the network. These vulnerabilities should be fixed as soon as possible.

The "Latest Fixed Version" column contains the latest version of the software that fixes the vulnerability.

The time estimation is based on the remediation type of the vulnerability. The time estimation is only a rough estimate for the first fix. The time estimation for the following fixes is usually much lower.
Vulnerabilities listed in one section are not listed in the following sections. So if a vulnerability has a high priority but an update is available, it will only be listed in the missing updates section.

Results

Total Hosts	Total Vulnerabilities	Total Priority	Total Severity	Average Vulnerabilities per Host	Average Priority per Host
63	197	705.89	770.2	3.13	11.2

Hosts

Use the Searchbar to filter the vulnerabilities for IP Addresses.

Search for ip... Search

IP	Host Priority	Minor Updates	Major Updates	High Priority Vulnerabilities	Frequently Occuring Vulnerabilities	High Effort Remediations
127.0.0.1	14.8	1		1	2	
127.0.0.2	12.88			1	2	
127.0.0.3	12.0			1	2	
127.0.0.4	11.92				2	
127.0.0.5	7.35			1	2	

Minor Updates

IP	Host Name	Vulnerability Description	Port	Maximum Priority	Occurences	Oldest Installed Version	Latest Fixed Version	Solutions	CVEs	References	Time Estimation
127.0.0.1	example.com	Grafana < 8.5.21, 9.2.x < 9.2.13, 9.3.x < 9.3.8 Multiple Vulnerabilities	3000/tcp	14.8	6	9.3.2	9.3.8	Update to version 8.5.21, 9.2.13, 9.3.8 or later. Update to version 9.2.10, 9.3.4 or later. Update to version 9.2.13, 9.3.8 or later. Update to version 8.5.22, 9.2.15, 9.3.11, 9.4.7 or later.	CVE-2023-0507, CVE-2023-0594, CVE-2022-23552, CVE-2022-23498, CVE-2022-39324, CVE-2023-22462, CVE-2023-1410,	https://grafana.com/blog/2023/02/28/grafana-security-release-new-versions-with-security-fixes-for-cve-2023-0594-cve-2023-0507-and-cve-2023-22462/ https://github.com/grafana/grafana/security/advisories/GHSA-8xmm-x63g-j6xv https://grafana.com/blog/2023/01/25/grafana-security-releases-new-versions-with-fixes-for-cve-2022-23552-cve-2022-41912-and-cve-2022-39324/ https://github.com/grafana/grafana/security/advisories/GHSA-2j8f-6whh-frc8 https://github.com/grafana/grafana/security/advisories/GHSA-4724-7pwc-3jpw https://github.com/grafana/grafana/security/advisories/GHSA-7rqg-hjwc-6mjf https://github.com/grafana/bugbounty/security/advisories/GHSA-qrrg-gw7w-vp76 https://grafana.com/blog/2023/03/22/grafana-security-release-new-versions-with-security-fixes-for-cve-2023-1410/	1.0

Major Updates

No Major Updates found.

High Priority Vulnerabilities (No Updates Available)

IP	Host Name	Vulnerability Description	Port	Maximum Priority	Occurences	Oldest Installed Version	Latest Fixed Version	Solutions	CVEs	References	Time Estimation
127.0.0.2	example2.com	Test HTTP dangerous methods	4447/tcp	12.38	2	None	EOL or Version Independent Problem, see	Use access restrictions to these dangerous	None	http://www.securityfocus.com/bid/12141	2.0

B

Code

B.1. Script: split_subnets.sh

```
#!/bin/bash
usage() {
        echo "usage: $(basename "$0") <file>"
        exit 1
}
if [  $# -ne 1 ]; then
        usage
fi
if [ ! -f "$1" ]; then
    echo "File $1 does not exist"
    exit 1
fi

mkdir -p subnets
cd subnets

SUBNET=()
NRHOSTS=0
MAXHOSTS=4096

while read -r line; do
    SUBNETSIZE=$(ipcalc "$line" | grep "Hosts/Net:" | cut -d " " -f 2)
```

```
    echo "$SUBNETSIZE"
    if [ "$(($NRHOSTS + $SUBNETSIZE))" -gt "$MAXHOSTS" ]; then
        FILENAME="${SUBNET[0]}-${SUBNET[${#SUBNET[@]}-1]}.txt"
        touch ${FILENAME//\//_}
        printf '%s\n' "${SUBNET[@]}" >> ${FILENAME//\//_}
        SUBNET=()
        NRHOSTS=0
    fi
    NRHOSTS=$(($NRHOSTS + $SUBNETSIZE))
    SUBNET+=("$line")
done < "../$1"

if [ ${#SUBNET[@]} -gt 0 ]; then
    FILENAME="${SUBNET[0]}-${SUBNET[${#SUBNET[@]}-1]}.txt"
    touch ${FILENAME//\//_}
    printf '%s\n' "${SUBNET[@]}" >> ${FILENAME//\//_}
fi
cd ..
exit 0
```

B.2. SQL Query: Major Updates

```sql
CREATE TABLE missing_updates AS
WITH tmp_cves AS (
    SELECT
        r.id as result_id,
        rtrim(replace(group_concat(DISTINCT c.cve||'@!'), '@!,', x'0a'),'@!')
            as "cves"
    FROM results as r LEFT JOIN cves as c ON r.id = c.result_id
    GROUP BY r.vulnerable_service, r.ip
),
tmp_solution_urls AS (
    SELECT
        r.id as result_id,
        rtrim(replace(group_concat(DISTINCT s.url||'@!'), '@!,', x'0a'),'@!')
            as solutions
    FROM results as r LEFT JOIN solution_urls as s ON r.id = s.result_id
    GROUP BY r.vulnerable_service, r.ip
)
SELECT
    r.faculty as faculty,
    r.ip as ip,
    r.hostname as host_name,
    r.vulnerable_service as service,
    r.name as name,
    r.port as port,
    ROUND(MAX(r.priority), 2) as max_priority,
    ROUND(SUM(r.priority), 2) as total_priority,
    ROUND(AVG(r.priority), 2) as avg_priority,
    COUNT(r.id) as occurences,
    rtrim(
        replace(group_concat(DISTINCT r.solution_text||'@!'), '@!,', x'0a'),'@!'
    ) as solution_text,
    MIN(r.vulnerable_version) as installed_version,
    MAX(r.fixed_version) as fixed_version,
    MAX(c.cves)  as "cves",
    MAX(s.solutions) as solutions,
    MAX(r.exploit_exists) as exploit_exists,
    MAX(r.complexity) as cost,
    r.severity as severity,
    sqrt(SUM(pow(r.severity, 2)/r.complexity)) as cvss_per_hour
```

```
FROM results as r
    LEFT JOIN tmp_cves as c ON r.id = c.result_id
    LEFT JOIN tmp_solution_urls as s ON r.id = s.result_id
WHERE r.processed = 0
    AND r.solution_type = "VendorFix"
    AND (
        LOWER(r.solution_text) LIKE "update %"
        OR LOWER(r.solution_text) LIKE "upgrade %"
    )
GROUP BY r.ip, r.vulnerable_service, r.port
ORDER BY faculty, exploit_exists DESC, max_priority DESC;
```

B.3. HTML template: output_template.html.j2

```
<!DOCTYPE html>
<html>

<head>
    <title>Vulnerability Report</title>
</head>

<body>
    <h1>Vulnerability Report</h1>
    <h2>Reading the Report</h2>
    <p>
        This report is automatically generated from the results of a vulnerability
        scan report.<br>
        It is divided into six sections per faculty. <br>
        The first section contains general statistics about each faculties' security
        status.<br>
        The second section contains a list of vulnerabilities that can be
        remidiated by installing minor updates.<br>
        The third section contains a list of vulnerabilities that can be
        remidiated by installing missing updates.<br>
        The fourth section contains a list of vulnerabilities that have a high
        priority, either because they are
        exploitable or because they have a high priority score.<br>
        The fifth section contains a list of vulnerabilities that are widely spread
        across the network. Usually these
```

```
26          vulnerabilities are caused by a misconfiguration of the network or a widely
27          used service.<br>
28          The sixth section contains a list of vulnerabilities that are neither
29          missing updates, high priority nor widely
30          spread, but are expected to require a lot of time to fix.<br>
31          <br>
32          The vulnerabilities are sorted by priority. So the report should be read
33          from top to bottom.
34          <br>
35          Vulnerabilities that are marked with a red background color have a publicly
36          available exploit or a priority
37          score of 10 or higher.
38          A maximum priority of above 10 is usually caused by vulnerabilities like
39          Remote Code Execution, Default
40          Credentials or other vulnerabilities that, when exploited, can cause a lot
41          of damage to the network.
42          These vulnerabilities should be fixed as soon as possible.
43          <br>
44          <br>
45          The "Latest Fixed Version" column contains the latest version of the software
46          that fixes the vulnerability.
47          <br>
48          <br>
49          The time estimation is based on the remediation type of the vulnerability.
50          The time estimation is only a rough
51          estimate for the first fix. The time estimation for the following fixes is
52          usually much lower.
53          <br>
54          Vulnerabilities listed in one section are not listed in the following
55          sections. So if a vulnerability has a high
56          priority but an update is available, it will only be listed in the missing
57          updates section.
58      </p>
59      <hr />
60      <hr />
61      {% for faculty in items.faculties %}
62      <h2>Faculty: {{ faculty }}</h2>
63      <table>
64          <thead>
65              <tr>
66                  <th>Total Hosts</th>
67                  <th>Total Vulnerabilities</th>
68                  <th>Total Priority</th>
69                  <th>Total Severity</th>
```

```
70                  <th>Average Vulnerabilities per Host</th>
71                  <th>Average Priority per Host</th>
72              </tr>
73          </thead>
74          <tbody>
75              <tr>
76                  <td>{{ items.results[faculty].vuln_stats.total_hosts }}</td>
77                  <td>{{ items.results[faculty].vuln_stats.total_vulns }}</td>
78                  <td>{{ items.results[faculty].vuln_stats.total_priority }}</td>
79                  <td>{{ items.results[faculty].vuln_stats.total_severity }}</td>
80                  <td>{{ items.results[faculty].vuln_stats.avg_vulns }}</td>
81                  <td>{{ items.results[faculty].vuln_stats.avg_priority }}</td>
82              </tr>
83          </tbody>
84      </table>
85      <h3>Minor Updates</h3>
86      {% if items.results[faculty].minor_updates|length != 0 %}
87      <table>
88          <thead>
89              <tr>
90                  <th>IP</th>
91                  <th>Host Name</th>
92                  <th>Vulnerability Description</th>
93                  <th>Port</th>
94                  <th>Maximum Priority</th>
95                  <th>Occurences</th>
96                  <th>Oldest Installed Version</th>
97                  <th>Latest Fixed Version</th>
98                  <th>Solutions</th>
99                  <th>CVEs</th>
100                 <th>References</th>
101                 <th>Time Estimation</th>
102             </tr>
103         </thead>
104         <tbody>
105             {% for vuln in items.results[faculty].minor_updates %}
106             <tr {% if vuln.exploit_exists or vuln.max_priority>= 10.0 %}
107                 style="background-color:#ff0000;" {% endif %}>
108                 <td>{% for ip in vuln.host_ips %}{{ ip }}, {% endfor %}</td>
109                 <td>{% for host_name in vuln.host_names %}{{ host_name }},
110                     {% endfor %}</td>
111                 <td>{{ vuln.vuln_description }}</td>
112                 <td>{% for port in vuln.port %}{{ port }}, {% endfor %}</td>
113                 <td>{{ vuln.max_priority }}</td>
```

```
114                 <td>{{ vuln.vuln_occurrences }}</td>
115                 <td>{{ vuln.oldest_installed_version }}</td>
116                 <td>{{ vuln.latest_fixed_version }}</td>
117                 <td>{% for solution in vuln.solution_texts %}{{ solution }} <br>
118                     {% endfor %}</td>
119                 <td>{% for cve in vuln.cves %}{{ cve }}, {% endfor %} </td>
120                 <td>{% for reference in vuln.solution_urls %}
121                     <a href="{{ reference }}">{{ reference }}</a> <br>{% endfor %}
122                 </td>
123                 <td>{{ vuln.effort }}</td>
124             </tr>
125         </tbody>
126         {% endfor %}
127     </table>
128     {% else %}
129     <p>No Minor Updates found.</p>
130     {% endif %}
131     <h3>Major Updates</h3>
132     {% if items.results[faculty].missing_updates|length != 0 %}
133     <table>
134         <thead>
135             <tr>
136                 <th>IP</th>
137                 <th>Host Name</th>
138                 <th>Vulnerability Description</th>
139                 <th>Port</th>
140                 <th>Maximum Priority</th>
141                 <th>Occurences</th>
142                 <th>Oldest Installed Version</th>
143                 <th>Latest Fixed Version</th>
144                 <th>Solutions</th>
145                 <th>CVEs</th>
146                 <th>References</th>
147                 <th>Time Estimation</th>
148             </tr>
149         </thead>
150         <tbody>
151             {% for vuln in items.results[faculty].missing_updates %}
152             <tr {% if vuln.exploit_exists or vuln.max_priority>= 10.0 %}
153                 style="background-color:#ff0000;" {% endif %}>
154                 <td>{% for ip in vuln.host_ips %}{{ ip }}, {% endfor %}</td>
155                 <td>{% for host_name in vuln.host_names %}{{ host_name }},
156                     {% endfor %}</td>
157                 <td>{{ vuln.vuln_description }}</td>
```

```
158                 <td>{% for port in vuln.port %}{{ port }}, {% endfor %}</td>
159                 <td>{{ vuln.max_priority }}</td>
160                 <td>{{ vuln.vuln_occurrences }}</td>
161                 <td>{{ vuln.oldest_installed_version }}</td>
162                 <td>{{ vuln.latest_fixed_version }}</td>
163                 <td>{% for solution in vuln.solution_texts %}{{ solution }} <br>
164                     {% endfor %}</td>
165                 <td>{% for cve in vuln.cves %}{{ cve }}, {% endfor %} </td>
166                 <td>{% for reference in vuln.solution_urls %}
167                     <a href="{{ reference }}">{{ reference }}</a> <br>{% endfor %}
168                 </td>
169                 <td>{{ vuln.effort }}</td>
170             </tr>
171         </tbody>
172         {% endfor %}
173     </table>
174     {% else %}
175     <p>No Major Updates found.</p>
176     {% endif %}
177     <h3>High Priority Vulnerabilities (No Updates Available)</h3>
178     {% if items.results[faculty].high_priority_vulnerabilities |length != 0 %}
179     <table>
180         <thead>
181             <tr>
182                 <th>IP</th>
183                 <th>Host Name</th>
184                 <th>Vulnerability Description</th>
185                 <th>Port</th>
186                 <th>Maximum Priority</th>
187                 <th>Occurences</th>
188                 <th>Oldest Installed Version</th>
189                 <th>Latest Fixed Version</th>
190                 <th>Solutions</th>
191                 <th>CVEs</th>
192                 <th>References</th>
193                 <th>Time Estimation</th>
194             </tr>
195         </thead>
196         <tbody>
197             {% for vuln in items.results[faculty].high_priority_vulnerabilities %}
198             <tr {% if vuln.exploit_exists or vuln.max_priority>= 10.0 %}
199                 style="background-color:#ff0000;" {% endif %}>
200                 <td>{% for ip in vuln.host_ips %}{{ ip }}, {% endfor %}</td>
201                 <td>{% for host_name in vuln.host_names %}{{ host_name }},
```

```
202                     {% endfor %}</td>
203                 <td>{{ vuln.vuln_description }}</td>
204                 <td>{% for port in vuln.port %}{{ port }}, {% endfor %}</td>
205                 <td>{{ vuln.max_priority }}</td>
206                 <td>{{ vuln.vuln_occurrences }}</td>
207                 <td>{{ vuln.oldest_installed_version }}</td>
208                 <td>{{ vuln.latest_fixed_version }}</td>
209                 <td>{% for solution in vuln.solution_texts %}{{ solution }}<br>
210                     {% endfor %}</td>
211                 <td>{% for cve in vuln.cves %}{{ cve }}, {% endfor %}</td>
212                 <td>{% for reference in vuln.solution_urls %}
213                     <a href="{{ reference }}">{{ reference }}</a><br>{% endfor %}
214                 </td>
215                 <td>{{ vuln.effort }}</td>
216             </tr>
217         </tbody>
218         {% endfor %}
219     </table>
220     {% else %}
221     <p>No High Priority Vulnerabilities found.</p>
222     {% endif %}
223     <h3>Frequently Occuring Vulnerabilities</h3>
224     {% if items.results[faculty].wide_spread_vulnerabilities |length != 0 %}
225     <table>
226         <thead>
227             <tr>
228                 <th>IP</th>
229                 <th>Host Name</th>
230                 <th>Vulnerability Description</th>
231                 <th>Port</th>
232                 <th>Maximum Priority</th>
233                 <th>Occurences</th>
234                 <th>Oldest Installed Version</th>
235                 <th>Latest Fixed Version</th>
236                 <th>Solutions</th>
237                 <th>CVEs</th>
238                 <th>References</th>
239                 <th>Time Estimation</th>
240             </tr>
241         </thead>
242         <tbody>
243             {% for vuln in items.results[faculty].wide_spread_vulnerabilities %}
244             <tr {% if vuln.exploit_exists or vuln.max_priority>= 10.0 %}
245                 style="background-color:#ff0000;" {% endif %}>
```

```
246                 <td>{% for ip in vuln.host_ips %}{{ ip }}, {% endfor %}</td>
247                 <td>{% for host_name in vuln.host_names %}{{ host_name }},
248                     {% endfor %}</td>
249                 <td>{{ vuln.vuln_description }}</td>
250                 <td>{% for port in vuln.port %}{{ port }}, {% endfor %}</td>
251                 <td>{{ vuln.max_priority }}</td>
252                 <td>{{ vuln.vuln_occurrences }}</td>
253                 <td>{{ vuln.oldest_installed_version }}</td>
254                 <td>{{ vuln.latest_fixed_version }}</td>
255                 <td>{% for solution in vuln.solution_texts %}{{ solution }}<br>
256                     {% endfor %}</td>
257                 <td>{% for cve in vuln.cves %}{{ cve }}, {% endfor %}</td>
258                 <td>{% for reference in vuln.solution_urls %}
259                     <a href="{{ reference }}">{{ reference }}</a><br>{% endfor %}
260                 </td>
261                 <td>{{ vuln.effort }}</td>
262             </tr>
263         </tbody>
264         {% endfor %}
265     </table>
266     {% else %}
267     <p>No Frequently Occuring Vulnerabilities found.</p>
268     {% endif %}
269     <h3>High Effort Remediations</h3>
270     {% if items.results[faculty].high_effort_remediations |length != 0 %}
271     <table>
272         <thead>
273             <tr>
274                 <th>IP</th>
275                 <th>Host Name</th>
276                 <th>Vulnerability Description</th>
277                 <th>Port</th>
278                 <th>Maximum Priority</th>
279                 <th>Occurences</th>
280                 <th>Oldest Installed Version</th>
281                 <th>Latest Fixed Version</th>
282                 <th>Solutions</th>
283                 <th>CVEs</th>
284                 <th>References</th>
285                 <th>Time Estimation</th>
286             </tr>
287         </thead>
288         <tbody>
289             {% for vuln in items.results[faculty].high_effort_remediations %}
```

```
290             <tr {% if vuln.exploit_exists or vuln.max_priority>= 10.0 %}
291                 style="background-color:#ff0000;" {% endif %}>
292                 <td>{% for ip in vuln.host_ips %}{{ ip }}, {% endfor %}</td>
293                 <td>{% for host_name in vuln.host_names %}{{ host_name }},
294                     {% endfor %}</td>
295                 <td>{{ vuln.vuln_description }}</td>
296                 <td>{% for port in vuln.port %}{{ port }}, {% endfor %}</td>
297                 <td>{{ vuln.max_priority }}</td>
298                 <td>{{ vuln.vuln_occurrences }}</td>
299                 <td>{{ vuln.oldest_installed_version }}</td>
300                 <td>{{ vuln.latest_fixed_version }}</td>
301                 <td>{% for solution in vuln.solution_texts %}{{ solution }}<br>
302                     {% endfor %}</td>
303                 <td>{% for cve in vuln.cves %}{{ cve }}, {% endfor %}</td>
304                 <td>{% for reference in vuln.solution_urls %}
305                     <a href="{{ reference }}">{{ reference }}</a><br>{% endfor %}
306                 </td>
307                 <td>{{ vuln.effort }}</td>
308             </tr>
309         </tbody>
310         {% endfor %}
311     </table>
312     {% else %}
313     <p>No High Effort Remediations found.</p>
314     {% endif %}
315 
316     <h3>Remaining Vulnerabilities</h3>
317     {% if items.results[faculty].high_effort_remediations | length == 0
318     and items.results[faculty].wide_spread_vulnerabilities |length == 0
319     and items.results[faculty].minor_updates|length == 0
320     and items.results[faculty].missing_updates|length == 0 %}
321     <table>
322         <thead>
323             <tr>
324                 <th>IP</th>
325                 <th>Host Name</th>
326                 <th>Vulnerability Description</th>
327                 <th>Port</th>
328                 <th>Maximum Priority</th>
329                 <th>Oldest Installed Version</th>
330                 <th>Latest Fixed Version</th>
331                 <th>Solutions</th>
332                 <th>CVEs</th>
333                 <th>References</th>
```

```
334                 <th>Time Estimation</th>
335             </tr>
336         </thead>
337         <tbody>
338             {% for vuln in items.results[faculty].remaining_vulnerabilities %}
339             <tr {% if vuln.exploit_exists or vuln.max_priority>= 10.0 %}
340                 style="background-color:#ff0000;" {% endif %}>
341                 <td>{% for ip in vuln.host_ips %}{{ ip }}, {% endfor %}</td>
342                 <td>{% for host_name in vuln.host_names %}{{ host_name }},
343                     {% endfor %}</td>
344                 <td>{{ vuln.vuln_description }}</td>
345                 <td>{% for port in vuln.port %}{{ port }}, {% endfor %}</td>
346                 <td>{{ vuln.max_priority }}</td>
347                 <td>{{ vuln.oldest_installed_version }}</td>
348                 <td>{{ vuln.latest_fixed_version }}</td>
349                 <td>{% for solution in vuln.solution_texts %}{{ solution }}<br>
350                     {% endfor %}</td>
351                 <td>{% for cve in vuln.cves %}{{ cve }}, {% endfor %}</td>
352                 <td>{% for reference in vuln.solution_urls %}
353                     <a href="{{ reference }}">{{ reference }}</a><br>{% endfor %}
354                 </td>
355                 <td>{{ vuln.effort }}</td>
356             </tr>
357         </tbody>
358         {% endfor %}
359
360     </table>
361     {% else %}
362     <p>There are {{ items.results[faculty].remaining_vulnerabilities | length }}
363         uncategorized vulnerabilities.<br>These will be available as soon as all
364         other vulnerabilities have been remediated.</p>
365     {% endif %}
366
367     {% endfor %}
368     <style>
369         table {
370             border: 1px solid black;
371         }
372
373         th,
374         td {
375             border-bottom: 1px solid black;
376             border-right: 1px dashed black;
377             horizontal-align: left;
```

```
        }

        table,
        th,
        tr,
        td {
            border-collapse: collapse;
            vertical-align: top;
        }
    </style>

</body>


</html>
```

B.4. HTML template: single_faculty_report.html.j2

```
<!DOCTYPE html>
<html>

<head>
    <title>Vulnerability Report</title>
</head>

<body>
    <h1>Vulnerability Report for faculty: {{faculty}}</h1>

    <h2>Reading the Report</h2>
    <p>
        This report is automatically generated from the results of a vulnerability
        scan report.<br>
        It is divided into six sections. <br>
        The first section contains general statistics about the faculties' security
        status.<br>
        The second section contains a list of vulnerabilities that can be
        remidiated by installing minor updates.<br>
        The third section contains a list of vulnerabilities that can be
        remidiated by installing missing updates.<br>
```

```
22          The fourth section contains a list of vulnerabilities that have a high
23          priority, either because they are
24          exploitable or because they have a high priority score.<br>
25          The fifth section contains a list of vulnerabilities that are widely spread
26          across the network. Usually these
27          vulnerabilities are caused by a misconfiguration of the network or a widely
28          used service.<br>
29          The sixth section contains a list of vulnerabilities that are neither
30          missing updates, high priority nor widely
31          spread, but are expected to require a lot of time to fix.<br>
32          <br>
33          The vulnerabilities are sorted by priority. So the report should be read
34          from top to bottom.
35          <br>
36          Vulnerabilities that are marked with a red background color have a publicly
37          available exploit or a priority
38          score of 10 or higher.
39          A maximum priority of above 10 is usually caused by vulnerabilities like
40          Remote Code Execution, Default
41          Credentials or other vulnerabilities that, when exploited, can cause a lot
42          of damage to the network.
43          These vulnerabilities should be fixed as soon as possible.
44          <br>
45          <br>
46          The "Latest Fixed Version" column contains the latest version of the software
47          that fixes the vulnerability.
48          <br>
49          <br>
50          The time estimation is based on the remediation type of the vulnerability.
51          The time estimation is only a rough
52          estimate for the first fix. The time estimation for the following fixes is
53          usually much lower.
54          <br>
55          Vulnerabilities listed in one section are not listed in the following
56          sections. So if a vulnerability has a high
57          priority but an update is available, it will only be listed in the missing
58          updates section.
59      </p>
60      <hr />
61      <hr />
62
63
64      <h2>Results</h2>
65      <table>
```

```
66          <thead>
67              <tr>
68                  <th>Total Hosts</th>
69                  <th>Total Vulnerabilities</th>
70                  <th>Total Priority</th>
71                  <th>Total Severity</th>
72                  <th>Average Vulnerabilities per Host</th>
73                  <th>Average Priority per Host</th>
74              </tr>
75          </thead>
76          <tbody>
77              <tr>
78                  <td>{{ results.vuln_stats.total_hosts }}</td>
79                  <td>{{ results.vuln_stats.total_vulns }}</td>
80                  <td>{{ results.vuln_stats.total_priority }}</td>
81                  <td>{{ results.vuln_stats.total_severity }}</td>
82                  <td>{{ results.vuln_stats.avg_vulns }}</td>
83                  <td>{{ results.vuln_stats.avg_priority }}</td>
84              </tr>
85          </tbody>
86      </table>
87      <h2>Hosts</h2>
88      <p>Use the Searchbar to filter the vulnerabilities for IP Addresses.</p>
89      <input type="text" id="searchbar" placeholder="Search for ip...">
90      <button onclick="search()">Search</button>
91      <br>
92      <table>
93          <thead>
94              <tr>
95                  <th>IP</th>
96                  <th>Host Priority</th>
97                  <th>Minor Updates</th>
98                  <th>Major Updates</th>
99                  <th>High Priority Vulnerabilities</th>
100                 <th>Frequently Occuring Vulnerabilities</th>
101                 <th>High Effort Remediations</th>
102             </tr>
103         </thead>
104         <tbody>
105             {% for ip in results.ips %}
106             <tr {% if ip[1].exploit_exists or ip[1].max_priority>= 10.0 %}
107                 style="background-color:#ff0000;" {% endif %}>
108                 <td>{{ ip[0] }}</td>
109                 <td>{{ ip[1].max_priority }}</td>
```

```
110                 <td>{{ ip[1].minor_updates }}</td>
111                 <td>{{ ip[1].missing_updates }}</td>
112                 <td>{{ ip[1].high_priority_vulnerabilities }}</td>
113                 <td>{{ ip[1].wide_spread_vulnerabilities }}</td>
114                 <td>{{ ip[1].high_effort_remediations }}</td>
115             </tr>
116             {% endfor %}
117     </table>
118
119     <h3>Minor Updates</h3>
120     {% if results.minor_updates|length != 0 %}
121     <table>
122         <thead>
123             <tr>
124                 <th>IP</th>
125                 <th>Host Name</th>
126                 <th>Vulnerability Description</th>
127                 <th>Port</th>
128                 <th>Maximum Priority</th>
129                 <th>Occurences</th>
130                 <th>Oldest Installed Version</th>
131                 <th>Latest Fixed Version</th>
132                 <th>Solutions</th>
133                 <th>CVEs</th>
134                 <th>References</th>
135                 <th>Time Estimation</th>
136             </tr>
137         </thead>
138         <tbody>
139             {% for vuln in results.minor_updates %}
140             <tr class="results" {% if vuln.exploit_exists or vuln.max_priority>= 10.0 %}
141                 style="background-color:#ff0000;" {% endif %}>
142                 <td id="ips">{% for ip in vuln.host_ips %}{{ ip }}, {% endfor %}</td>
143                 <td>{% for host_name in vuln.host_names %}{{ host_name }},
144                     {% endfor %}
145                 </td>
146                 <td>{{ vuln.vuln_description }}</td>
147                 <td>{% for port in vuln.port %}{{ port }}, {% endfor %}</td>
148                 <td>{{ vuln.max_priority }}</td>
149                 <td>{{ vuln.vuln_occurrences }}</td>
150                 <td>{{ vuln.oldest_installed_version }}</td>
151                 <td>{{ vuln.latest_fixed_version }}</td>
152                 <td>{% for solution in vuln.solution_texts %}{{ solution }} <br>
153                     {% endfor %}
```

```
154                 </td>
155                 <td>{% for cve in vuln.cves %}{{ cve }}, {% endfor %} </td>
156                 <td>{% for reference in vuln.solution_urls %}
157                     <a href="{{ reference }}">{{ reference }}</a> <br>
158                     {% endfor %}
159                 </td>
160                 <td>{{ vuln.effort }}</td>
161             </tr>
162         </tbody>
163         {% endfor %}
164 
165     </table>
166     {% else %}
167     <p>No Minor Updates found.</p>
168     {% endif %}
169 
170     <h3>Major Updates</h3>
171     {% if results.missing_updates|length != 0 %}
172     <table>
173         <thead>
174             <tr>
175                 <th>IP</th>
176                 <th>Host Name</th>
177                 <th>Vulnerability Description</th>
178                 <th>Port</th>
179                 <th>Maximum Priority</th>
180                 <th>Occurences</th>
181                 <th>Oldest Installed Version</th>
182                 <th>Latest Fixed Version</th>
183                 <th>Solutions</th>
184                 <th>CVEs</th>
185                 <th>References</th>
186                 <th>Time Estimation</th>
187             </tr>
188         </thead>
189         <tbody>
190             {% for vuln in results.missing_updates %}
191             <tr class="results" {% if vuln.exploit_exists or vuln.max_priority>= 10.0 %}
192                 style="background-color:#ff0000;" {% endif %}>
193                 <td id="ips">{% for ip in vuln.host_ips %}{{ ip }}, {% endfor %}</td>
194                 <td>{% for host_name in vuln.host_names %}{{ host_name }},
195                     {% endfor %}
196                 </td>
197                 <td>{{ vuln.vuln_description }}</td>
```

```
198                 <td>{% for port in vuln.port %}{{ port }}, {% endfor %}</td>
199                 <td>{{ vuln.max_priority }}</td>
200                 <td>{{ vuln.vuln_occurrences }}</td>
201                 <td>{{ vuln.oldest_installed_version }}</td>
202                 <td>{{ vuln.latest_fixed_version }}</td>
203                 <td>{% for solution in vuln.solution_texts %}{{ solution }} <br>
204                     {% endfor %}
205                 </td>
206                 <td>{% for cve in vuln.cves %}{{ cve }}, {% endfor %} </td>
207                 <td>{% for reference in vuln.solution_urls %}
208                     <a href="{{ reference }}">{{ reference }}</a> <br>
209                     {% endfor %}
210                 </td>
211                 <td>{{ vuln.effort }}</td>
212             </tr>
213         </tbody>
214         {% endfor %}
215
216     </table>
217     {% else %}
218     <p>No Major Updates found.</p>
219     {% endif %}
220
221     <h3>High Priority Vulnerabilities (No Updates Available)</h3>
222     {% if results.high_priority_vulnerabilities |length != 0 %}
223     <table>
224         <thead>
225             <tr>
226                 <th>IP</th>
227                 <th>Host Name</th>
228                 <th>Vulnerability Description</th>
229                 <th>Port</th>
230                 <th>Maximum Priority</th>
231                 <th>Occurences</th>
232                 <th>Oldest Installed Version</th>
233                 <th>Latest Fixed Version</th>
234                 <th>Solutions</th>
235                 <th>CVEs</th>
236                 <th>References</th>
237                 <th>Time Estimation</th>
238             </tr>
239         </thead>
240         <tbody>
241             {% for vuln in results.high_priority_vulnerabilities %}
```

```
242             <tr class="results" {% if vuln.exploit_exists or vuln.max_priority>= 10.0 %}
243                 style="background-color:#ff0000;" {% endif %}>
244                 <td id="ips">{% for ip in vuln.host_ips %}{{ ip }}, {% endfor %}</td>
245                 <td>{% for host_name in vuln.host_names %}{{ host_name }},
246                     {% endfor %}</td>
247                 <td>{{ vuln.vuln_description }}</td>
248                 <td>{% for port in vuln.port %}{{ port }}, {% endfor %}</td>
249                 <td>{{ vuln.max_priority }}</td>
250                 <td>{{ vuln.vuln_occurrences }}</td>
251                 <td>{{ vuln.oldest_installed_version }}</td>
252                 <td>{{ vuln.latest_fixed_version }}</td>
253                 <td>{% for solution in vuln.solution_texts %}{{ solution }}<br>
254                     {% endfor %}
255                 </td>
256                 <td>{% for cve in vuln.cves %}{{ cve }}, {% endfor %}</td>
257                 <td>{% for reference in vuln.solution_urls %}
258                     <a href="{{ reference }}">{{ reference }}</a><br>
259                     {% endfor %}
260                 </td>
261                 <td>{{ vuln.effort }}</td>
262             </tr>
263         </tbody>
264         {% endfor %}
265
266     </table>
267     {% else %}
268     <p>No High Priority Vulnerabilities found.</p>
269     {% endif %}
270
271     <h3>Frequently Occuring Vulnerabilities</h3>
272     {% if results.wide_spread_vulnerabilities |length != 0 %}
273     <table>
274         <thead>
275             <tr>
276                 <th>IP</th>
277                 <th>Host Name</th>
278                 <th>Vulnerability Description</th>
279                 <th>Port</th>
280                 <th>Maximum Priority</th>
281                 <th>Occurences</th>
282                 <th>Oldest Installed Version</th>
283                 <th>Latest Fixed Version</th>
284                 <th>Solutions</th>
285                 <th>CVEs</th>
```

```
286                 <th>References</th>
287                 <th>Time Estimation</th>
288             </tr>
289         </thead>
290         <tbody>
291             {% for vuln in results.wide_spread_vulnerabilities %}
292             <tr class="results" {% if vuln.exploit_exists or vuln.max_priority>= 10.0 %}
293                 style="background-color:#ff0000;" {% endif %}>
294                 <td id="ips">{% for ip in vuln.host_ips %}{{ ip }}, {% endfor %}</td>
295                 <td>{% for host_name in vuln.host_names %}{{ host_name }},
296                     {% endfor %}
297                 </td>
298                 <td>{{ vuln.vuln_description }}</td>
299                 <td>{% for port in vuln.port %}{{ port }}, {% endfor %}</td>
300                 <td>{{ vuln.max_priority }}</td>
301                 <td>{{ vuln.vuln_occurrences }}</td>
302                 <td>{{ vuln.oldest_installed_version }}</td>
303                 <td>{{ vuln.latest_fixed_version }}</td>
304                 <td>{% for solution in vuln.solution_texts %}{{ solution }}<br>
305                     {% endfor %}
306                 </td>
307                 <td>{% for cve in vuln.cves %}{{ cve }}, {% endfor %}</td>
308                 <td>{% for reference in vuln.solution_urls %}
309                     <a href="{{ reference }}">{{ reference }}</a><br>
310                     {% endfor %}
311                 </td>
312                 <td>{{ vuln.effort }}</td>
313             </tr>
314         </tbody>
315         {% endfor %}
316
317     </table>
318     {% else %}
319     <p>No Frequently Occuring Vulnerabilities found.</p>
320     {% endif %}
321
322     <h3>High Effort Remediations</h3>
323     {% if results.high_effort_remediations |length != 0 %}
324     <table>
325         <thead>
326             <tr>
327                 <th>IP</th>
328                 <th>Host Name</th>
329                 <th>Vulnerability Description</th>
```

```
330                 <th>Port</th>
331                 <th>Maximum Priority</th>
332                 <th>Occurences</th>
333                 <th>Oldest Installed Version</th>
334                 <th>Latest Fixed Version</th>
335                 <th>Solutions</th>
336                 <th>CVEs</th>
337                 <th>References</th>
338                 <th>Time Estimation</th>
339             </tr>
340         </thead>
341         <tbody>
342             {% for vuln in results.high_effort_remediations %}
343             <tr class="results" {% if vuln.exploit_exists or vuln.max_priority>= 10.0 %}
344                 style="background-color:#ff0000;" {% endif %}>
345                 <td id="ips">{% for ip in vuln.host_ips %}{{ ip }}, {% endfor %}</td>
346                 <td>{% for host_name in vuln.host_names %}{{ host_name }},
347                     {% endfor %}</td>
348                 <td>{{ vuln.vuln_description }}</td>
349                 <td>{% for port in vuln.port %}{{ port }}, {% endfor %}</td>
350                 <td>{{ vuln.max_priority }}</td>
351                 <td>{{ vuln.vuln_occurrences }}</td>
352                 <td>{{ vuln.oldest_installed_version }}</td>
353                 <td>{{ vuln.latest_fixed_version }}</td>
354                 <td>{% for solution in vuln.solution_texts %}{{ solution }}<br>
355                     {% endfor %}
356                 </td>
357                 <td>{% for cve in vuln.cves %}{{ cve }}, {% endfor %}</td>
358                 <td>{% for reference in vuln.solution_urls %}
359                     <a href="{{ reference }}">{{ reference }}</a><br>{% endfor %}
360                 </td>
361                 <td>{{ vuln.effort }}</td>
362             </tr>
363         </tbody>
364         {% endfor %}
365
366     </table>
367     {% else %}
368     <p>No High Effort Remediations found.</p>
369     {% endif %}
370
371     <h3>Remaining Vulnerabilities</h3>
372     {% if results.high_effort_remediations | length == 0
373     and results.wide_spread_vulnerabilities |length == 0
```

```
374     and results.minor_updates|length == 0
375     and results.missing_updates|length == 0 %}
376     <table>
377         <thead>
378             <tr>
379                 <th>IP</th>
380                 <th>Host Name</th>
381                 <th>Vulnerability Description</th>
382                 <th>Port</th>
383                 <th>Maximum Priority</th>
384                 <th>Oldest Installed Version</th>
385                 <th>Latest Fixed Version</th>
386                 <th>Solutions</th>
387                 <th>CVEs</th>
388                 <th>References</th>
389                 <th>Time Estimation</th>
390             </tr>
391         </thead>
392         <tbody>
393             {% for vuln in results.remaining_vulnerabilities %}
394             <tr class="results" {% if vuln.exploit_exists or vuln.max_priority>= 10.0 %}
395                 style="background-color:#ff0000;" {% endif %}>
396                 <td id="ips">{% for ip in vuln.host_ips %}{{ ip }}, {% endfor %}</td>
397                 <td>{% for host_name in vuln.host_names %}{{ host_name }},
398                     {% endfor %}
399                 </td>
400                 <td>{{ vuln.vuln_description }}</td>
401                 <td>{% for port in vuln.port %}{{ port }}, {% endfor %}</td>
402                 <td>{{ vuln.max_priority }}</td>
403                 <td>{{ vuln.oldest_installed_version }}</td>
404                 <td>{{ vuln.latest_fixed_version }}</td>
405                 <td>{% for solution in vuln.solution_texts %}{{ solution }}<br>
406                     {% endfor %}
407                 </td>
408                 <td>{% for cve in vuln.cves %}{{ cve }}, {% endfor %}</td>
409                 <td>{% for reference in vuln.solution_urls %}
410                     <a href="{{ reference }}">{{ reference }}</a><br>
411                     {% endfor %}
412                 </td>
413                 <td>{{ vuln.effort }}</td>
414             </tr>
415         </tbody>
416         {% endfor %}
417
```

```
418     </table>
419     {% else %}
420     <p>There are {{ results.remaining_vulnerabilities | length }} uncategorized
421         vulnerabilities.<br>These will be available as soon as all other
422         vulnerabilities have been remediated.</p>
423     {% endif %}
424
425     <style>
426         table {
427             border: 1px solid black;
428         }
429
430         th,
431         td {
432             border-bottom: 1px solid black;
433             border-right: 1px dashed black;
434             horizontal-align: left;
435         }
436
437         table,
438         th,
439         tr,
440         td {
441             border-collapse: collapse;
442             vertical-align: top;
443         }
444     </style>
445
446 </body>
447
448 <script>
449     function search() {
450         const input = document.getElementById( 'searchbar' );
451         const filter = input.value.toUpperCase();
452         const results = document.getElementsByClassName( "results" );
453         for ( let i = 0; i < results.length; i++ ) {
454             let ips = results[ i ].querySelector( "#ips" )
455             if ( ips.innerHTML.toUpperCase().indexOf( filter ) > -1 || filter == "" ) {
456                 results[ i ].style.display = "";
457             } else {
458                 results[ i ].style.display = "none";
459             }
460         }
461     }
```

```
</script>


</html>
```

B.5. Script: simplify.py

```
from functools import reduce
import xml.etree.ElementTree as ET
from os import listdir
from os import mkdir
from os.path import isfile, join, isdir
import sqlite3
import subprocess
import ipaddress
import re
import sys
import jinja2
from send_mail import send_mail
from datetime import datetime


# Result class to store all results retrieved from the database
class Results:
    faculties: list = []
    vuln_stats: list = []
    vuln_stats_by_faculty: list = []
    highest_impact_remediations: list = []
    major_updates: list = []
    minor_updates: list = []
    high_priority_vulnerabilities: list = []
    wide_spread_vulnerabilities: list = []
    high_effort_remediations: list = []
    remaining_vulnerabilities: list = []
    stats: list = []


# A list of high risk vulnerabilities, that need to be focused on
HIGH_RISK_VULNS = [
    ("remote", 10.0),
    ("code execution", 10.0),
    ("sqli", 10.0),
    ("local", 10.0),
    ("xss", 10.0),
    ("denial of service", 10.0),
    ("buffer overflow", 10.0),
```

```
40      ("end of life", 5.0),
41      ("file write", 5.0),
42      ("file deletion", 5.0),
43      ("file modification", 5.0),
44      ("dangerous methods", 5.0),
45      ("dangerous http method", 5.0),
46      ("default credentials", 10.0),
47      ("privilege escalation", 10.0),
48  ]
49  # High Priority Threshold defines which vulnerabilities are considered
50  # high-priority in the SQL queries
51  HIGH_PRIORITY_THRESHOLD = 7.0
52  # Wide Spread Threshold defines how many occurences a vulnerability has to have
53  # tobe considered wide-spread in the SQL queries
54  WIDE_SPREAD_THRESHOLD = 10.0
55  # Reoccurence Factor is used to adjust the cost of remediation for
56  # reoccuring vulnerabilities
57  REOCCURENCE_FACTOR = 0.2
58  # Complexities
59  VENDOR_FIX = 1.0
60  MITIGATION = 2.0
61  WORKAROUND = 4.0
62  NONE_AVAILABLE = 9.0
63  WILL_NOT_FIX = 10.0
64
65  TEMPLATEFILE = "output_template.html.j2"
66
67  # Create output directory with current date if it does not exist
68  OUTPUTDIR = f"output_files-{datetime.today().strftime('%Y-%m-%d')}"
69  if not isdir(OUTPUTDIR):
70      mkdir(OUTPUTDIR)
71  OUTPUTFILE = "output"
72
73  # Read map from subnets to faculties that were scanned
74  with open("faculties.txt", "r") as f:
75      subnets = f.readlines()
76  SUBNETS = [x.strip().split("|") for x in subnets]
77
78  # Read map from faculty to email address
79  with open("emails.txt", "r") as f:
80      emails = f.readlines()
81  EMAIL_ADDRESSES = dict([tuple(x.strip().split("|")) for x in emails])
82
83  # Establish dummy database connection
```

```
conn = sqlite3.connect(":memory:")


# Establish database connection, creates database if it does not exist
def connect_to_database():
    global conn
    conn = sqlite3.connect("results.db")


# Initializes the database, clears all tables.
def init_database():
    global conn
    conn.execute("DROP TABLE IF EXISTS results")
    conn.execute(
        """CREATE TABLE results
        (
            id text,
            name text,
            ip text,
            hostname text,
            port text,
            subnet text,
            severity real,
            solution_type text,
            solution_text text,
            qod number,
            faculty text,
            priority real,
            vulnerable_service text,
            vulnerable_version text,
            fixed_version text,
            complexity real,
            exploit_exists number,
            processed number
        )"""
    )
    conn.execute("DROP TABLE IF EXISTS cves")
    conn.execute(
        """CREATE TABLE IF NOT EXISTS cves (
            result_id text,
            cve text
        )"""
    )
    conn.execute("DROP TABLE IF EXISTS solution_urls")
```

```
    conn.execute(
        """CREATE TABLE IF NOT EXISTS solution_urls (
            result_id text,
            url text
        )"""
    )
    conn.execute("DROP TABLE IF EXISTS tags")
    conn.execute(
        """CREATE TABLE IF NOT EXISTS tags (
            result_id text,
            tag text
        )"""
    )
    conn.commit()


# Extracts the vulnerable version number from the vulnerability description
# returns None if no version is found
def get_vulnerable_version(description):
    vulnerable_version = None
    try:
        # Check if installed version is in description
        if "installed version" in description.lower():
            # extract installed version from description
            vulnerable_version = re.search(
                "installed version:[ ]+([^ ]*)\n", description.lower()
            ).group(1)
        # clean up version number (i.e. remove trailing text or version ranges)
        if "-" in vulnerable_version:
            vulnerable_version = vulnerable_version.split("-")[0]
        vulnerable_version = vulnerable_version.replace("/[^0-9.,]+/", "")
        return vulnerable_version
    except Exception:
        return vulnerable_version


# Extracts the fixed version number from the vulnerability description or solution text
# returns None if no version is found
def get_fixed_version(description, solution_text):
    fixed_version = None
    # Check if fixed version is in description
    try:
        if "fixed version" in description.lower():
            # extract fixed version from description
```

```
            fixed_version = re.search(
                "fixed version:[ ]+([^\n ]*)", description.lower()
            ).group(1)
        # check if "eol version" is in description and assign placeholder version
        if "eol version" in description.lower():
            fixed_version = "99.99.99"
    except Exception:
        pass
    if fixed_version is None:
        # check if "update to version" or "update to" is in solution text
        try:
            if "update to version" in solution_text.lower():
                # extract version to update to from solution text
                fixed_version = re.search(
                    "update to version[ ]+([^\n ]*),", solution_text.lower()
                ).group(1)
            elif "update to" in solution_text.lower():
                # extract version to update to from solution text
                fixed_version = re.search(
                    "update to[ ]+([^\n ]*),", solution_text.lower()
                ).group(1)
        except Exception:
            pass
    try:
        # clean up version number (i.e. remove trailing text or version ranges)
        if "-" in fixed_version:
            fixed_version = fixed_version.split("-")[0]
        if "/" in fixed_version:
            fixed_version = fixed_version.split("/")[0]
        fixed_version = fixed_version.replace("/[^0-9.,]+/", "")
    except Exception:
        pass
    return fixed_version


# Determine which subnet an IP belongs to
# returns the subnet or None if no subnet is found
def get_subnet(ip):
    try:
        for sub in SUBNETS:
            if ipaddress.ip_address(ip) in ipaddress.ip_network(sub[0]):
                return sub
    except Exception:
        return None
```

```python


# Get all CVEs from the vulnerability references
# returns a list of CVEs or an empty list if no CVEs are found
def get_cves(refs):
    cves = []
    if refs is not None:
        for ref in refs.findall("ref"):
            try:
                if ref.attrib["type"] == "cve":
                    cves.append(ref.attrib["id"])
            except Exception:
                pass
    return cves


# Get all solution URLs from the vulnerability references
# returns a list of URLs or an empty list if no URLs are found
def get_solution_urls(refs):
    solution_urls = []
    if refs is not None:
        for ref in refs.findall("ref"):
            try:
                if ref.attrib["type"] == "url":
                    solution_urls.append(ref.attrib["id"])
            except Exception:
                pass
    return solution_urls


# Determine the complexity of remediation for a solution type
def get_complexity(solution_type):
    match solution_type:
        case "VendorFix":
            return VENDOR_FIX
        case "Mitigation":
            return MITIGATION
        case "Workaround":
            return WORKAROUND
        case "NoneAvailable":
            return NONE_AVAILABLE
        case "WillNotFix":
            return WILL_NOT_FIX
        case _:
```

```
            return None


cves_with_exploits = {
    'unknown': False,
}


# Check searchsploit for exploits for a given CVE (very slow, run in parallel)
def get_existing_exploits_by_cve(cve):
    # check if cve has already been checked
    if cve in cves_with_exploits:
        return cves_with_exploits[cve]
    try:
        # run searchsploit in a subprocess for cve
        result = subprocess.run(
            ['searchsploit', '--cve', cve], stdout=subprocess.PIPE)
        # retrieve result from stdout and check if exploits were found
        result = result.stdout.decode('utf-8')
        # since searchsploit returns a table, check if "Exploits: No Results"
        # and "Shellcodes: No Results" are in the result
        # if both are in the result, no exploits were found
        if "Exploits: No Results" in result and "Shellcodes: No Results" in result:
            cves_with_exploits[cve] = False
            return False
        cves_with_exploits[cve] = True
        return True
    except Exception:
        return False


# Load all XML files in directory 'results' into the xml_data array
# returns the xml_data array
def load_xml_data():
    # get list of xml files in directory 'results'
    result_files = [
        f for f in listdir("results") if isfile(join("results", f)) and f.endswith(".xml")
    ]
    # if no xml files are found, exit
    if len(result_files) == 0:
        print("No XML files found in directory 'results'")
        exit(1)
    xml_data = []

```

```
    # for each xml file, load the <results> element into the xml_data array
    try:
        for result_file in result_files:
            f = ET.parse(join("results", result_file))
            tmp_root = f.getroot()
            results = tmp_root.find("report").find("results")
            for result in results.findall("result"):
                xml_data.append(result)
    except Exception:
        print("XML files could not be parsed")
        print("Please make sure all XML files can be parsed by the xPath query " +
              "'/report results/result'")
        print("If the XML files are not in the expected format, " +
              "please change the xPath query in the function 'load_xml_data'")
        exit(1)
    return xml_data


def determine_service(name):
    return name.split(" ")[0].replace(":", "")


def get_email(faculty):
    try:
        return EMAIL_ADDRESSES[faculty]
    except Exception:
        return None


def print_progress(i, total_results):
    print(f"Progress: {i+1}/{total_results} ({round(((i+1)/total_results)*100, 2)}%)",
          end="\r" if i < total_results else "\n")


def pre_process():
    xml_data = load_xml_data()
    total_results = len(xml_data)

    for result_elem in xml_data:
        # extract the relevant data from the XML
        # ==== IF THERE ARE CHANGES TO THE XML STRUCTURE OR PROBLEMS WHEN PARSING ====
        # ==== THIS PART MIGHT NEED TO BE CHANGED ====
        try:
            # get the result id
```

```
348                 result_id = result_elem.attrib["id"]
349                 # get the vulnerability name
350                 name = result_elem.find("name").text
351                 # get the host ip, hostname and port
352                 ip = result_elem.find("host").text.strip()
353                 host_name = result_elem.find("host").find("hostname").text
354                 port = result_elem.find("port").text
355                 # get the <nvt> element
356                 nvt = result_elem.find("nvt")
357                 # get the cvss score
358                 severity = float(result_elem.find("severity").text)
359                 # get the proposed solution type and text
360                 solution_type = nvt.find("solution").attrib["type"]
361                 solution_text = nvt.find("solution").text
362                 # get the vulnerability tags and references
363                 tags = nvt.find("tags").text.replace(
364                     "\n", "").replace("\t", "").split("|")
365                 refs = nvt.find("refs")
366                 # get the quality of detection
367                 qod = float(result_elem.find("qod").find("value").text) / 100.0
368                 # get the description
369                 if result_elem.find("description") is not None:
370                     description = result_elem.find("description").text
371                 else:
372                     description = ""
373             except Exception:
374                 print("XML files could not be parsed")
375                 print("Please make sure all XML files can be parsed by the xPath query " +
376                       "'/report results/result'")
377                 print("If the XML files are not in the expected format, please change the xPath query
378                       " in the function 'pre_process'")
379                 exit(1)
380             # ==== END OF PART ====
381
382             print_progress(xml_data.index(result_elem), total_results)
383
384             subnet = get_subnet(ip)
385             faculty = subnet[1]
386             subnet = subnet[0]
387
388             vulnerable_service = determine_service(name)
389
390             cves = get_cves(refs)
391
```

```
392         solution_urls = get_solution_urls(refs)
393
394         vulnerable_version = get_vulnerable_version(description)
395
396         fixed_version = get_fixed_version(description, solution_text)
397
398         # determine complexity
399         complexity = get_complexity(solution_type)
400
401         # determine if there are existing exploits
402         # multithreading is used to speed up the process since
403         # searchsploit can take a long time to run
404         do_exploits_exist = False
405         if len(cves) > 0:
406             do_exploits_exist = reduce((lambda x, y: x or y), map(
407                 get_existing_exploits_by_cve, cves))
408
409         # calculate priority
410         priority = 0.0
411
412         priority += severity
413
414         # take high risk vulns into account
415         for high_risk_vuln, risk in HIGH_RISK_VULNS:
416             if high_risk_vuln in name.lower():
417                 priority += risk
418
419         # adjust priority based on quality of detection
420         priority *= qod
421
422         priority = round(priority, 2)
423
424         # insert result into database
425         conn.execute(
426             "INSERT INTO results VALUES " +
427             "(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)",
428             (
429                 result_id,
430                 name,
431                 ip,
432                 host_name,
433                 port,
434                 subnet,
435                 severity,
```

```
436                 solution_type,
437                 solution_text,
438                 qod,
439                 faculty,
440                 priority,
441                 vulnerable_service,
442                 vulnerable_version,
443                 fixed_version,
444                 complexity,
445                 do_exploits_exist,
446                 0
447             ),
448         )
449         for cve in cves:
450             conn.execute("INSERT INTO cves VALUES (?, ?)",
451                          (result_id, cve))
452
453         for url in solution_urls:
454             conn.execute("INSERT INTO solution_urls VALUES (?, ?)",
455                          (result_id, url))
456
457         for tag in tags:
458             conn.execute("INSERT INTO tags VALUES (?, ?)",
459                          (result_id, tag))
460
461         conn.commit()
462
463
464 def analyze():
465
466     # reset processed flag
467     conn.execute(
468         """UPDATE results
469             SET processed = 0
470         """
471     )
472     conn.commit()
473
474     # create view for list of faculties
475     conn.execute(
476         """CREATE VIEW IF NOT EXISTS faculties AS
477             SELECT DISTINCT faculty
478             FROM results
479             ORDER BY faculty ASC
```

```
        """
    )

    # create view for vulnerability stats by faculty
    conn.execute("DROP VIEW IF EXISTS vuln_stats_by_faculty")
    conn.execute(
        """CREATE VIEW IF NOT EXISTS vuln_stats_by_faculty AS
        WITH tmp AS (
            SELECT
                r.faculty as faculty,
                COUNT(DISTINCT r.ip) as total_hosts,
                COUNT(DISTINCT r.id) as total_vulns,
                ROUND(SUM(r.priority),2) as total_priority,
                ROUND(SUM(r.severity),2) as total_severity
            FROM results as r
            GROUP BY faculty
        )
        SELECT
            tmp.faculty as faculty,
            tmp.total_hosts as total_hosts,
            tmp.total_vulns as total_vulns,
            tmp.total_priority as total_priority,
            tmp.total_severity as total_severity,
            ROUND((CAST(tmp.total_vulns as REAL) / CAST(tmp.total_hosts as REAL)), 2)
                as vulns_per_host,
            ROUND((tmp.total_priority / CAST(tmp.total_hosts as REAL)), 2)
                as priority_per_host
        FROM tmp
        GROUP BY faculty
        ORDER BY faculty ASC
        """
    )

    # create view for minor updates,
    # i.e. updates that only increment the last version number
    conn.execute("DROP TABLE IF EXISTS minor_updates")
    conn.execute(
        """CREATE TABLE minor_updates AS
        WITH tmp_cves AS (
            SELECT r.id as result_id,
            -- concatenate all cves into one string, uses newline as delimiter
            rtrim(replace(group_concat(DISTINCT c.cve||'@!'), '@!,', x'0a'),'@!') as "cves"
            FROM results as r LEFT JOIN cves as c ON r.id = c.result_id
            GROUP BY r.vulnerable_service, r.ip
```

```
        ),
        tmp_solution_urls AS (
            SELECT r.id as result_id,
            -- concatenate all solution urls into one string, uses newline as delimiter
            rtrim(replace(group_concat(DISTINCT s.url||'@!'), '@!,', x'0a'),'@!') as solutions
            FROM results as r LEFT JOIN solution_urls as s ON r.id = s.result_id
            GROUP BY r.vulnerable_service, r.ip
        )
        SELECT
            r.faculty as faculty,
            r.ip as ip,
            r.hostname as host_name,
            r.vulnerable_service as service,
            r.name as name,
            r.port as port,
            ROUND(MAX(r.priority), 2) as max_priority,
            ROUND(SUM(r.priority), 2) as total_priority,
            ROUND(AVG(r.priority), 2) as avg_priority,
            COUNT(r.id) as occurences,
            -- concatenate solution texts into one string, uses newline as delimiter
            rtrim(replace(group_concat(DISTINCT r.solution_text||'@!'), '@!,', x'0a'),'@!')
                as solution_text,
            MIN(r.vulnerable_version) as installed_version,
            MAX(r.fixed_version) as fixed_version,
            MAX(c.cves)  as "cves",
            MAX(s.solutions) as solutions,
            MAX(r.exploit_exists) as exploit_exists,
            MAX(r.complexity) as cost,
            r.severity as severity,
            -- calculate RMS of cvss per hour
            sqrt(SUM(pow(r.severity, 2)/r.complexity)) as cvss_per_hour
        FROM results as r LEFT JOIN tmp_cves as c ON r.id = c.result_id
            LEFT JOIN tmp_solution_urls as s ON r.id = s.result_id
        WHERE r.solution_type = "VendorFix"
            AND LOWER(r.solution_text) LIKE "update %"
        GROUP BY r.ip, r.vulnerable_service, r.port
        HAVING substr(installed_version, 0, instr(installed_version, ".")) =
            substr(fixed_version, 0, instr(fixed_version, "."))
        ORDER BY faculty, exploit_exists DESC, max_priority DESC
        """
    )
    conn.commit()
    # mark results as processed
    conn.execute(
```

```
        """UPDATE results as r
        SET processed = 1
        WHERE EXISTS (
            SELECT *
            FROM minor_updates as m
            WHERE r.ip = m.ip and r.vulnerable_service = m.service and r.port = m.port
        )
        """
    )
    conn.commit()

    # view minor_updates query for documentation
    conn.execute("DROP TABLE IF EXISTS major_updates")
    conn.execute(
        """CREATE TABLE major_updates AS
        WITH tmp_cves AS (
            SELECT r.id as result_id,
            rtrim(replace(group_concat(DISTINCT c.cve||'@!'), '@!,', x'0a'),'@!') as "cves"
            FROM results as r LEFT JOIN cves as c ON r.id = c.result_id
            GROUP BY r.vulnerable_service, r.ip
        ),
        tmp_solution_urls AS (
            SELECT r.id as result_id,
            rtrim(replace(group_concat(DISTINCT s.url||'@!'), '@!,', x'0a'),'@!') as solutions
            FROM results as r LEFT JOIN solution_urls as s ON r.id = s.result_id
            GROUP BY r.vulnerable_service, r.ip
        )
        SELECT
            r.faculty as faculty,
            r.ip as ip,
            r.hostname as host_name,
            r.vulnerable_service as service,
            r.name as name,
            r.port as port,
            ROUND(MAX(r.priority), 2) as max_priority,
            ROUND(SUM(r.priority), 2) as total_priority,
            ROUND(AVG(r.priority), 2) as avg_priority,
            COUNT(r.id) as occurences,
            rtrim(replace(group_concat(DISTINCT r.solution_text||'@!'), '@!,', x'0a'),'@!')
                as solution_text,
            MIN(r.vulnerable_version) as installed_version,
            MAX(r.fixed_version) as fixed_version,
            MAX(c.cves)  as "cves",
            MAX(s.solutions) as solutions,
```

```
            MAX(r.exploit_exists) as exploit_exists,
            MAX(r.complexity) as cost,
            r.severity as severity,
            sqrt(SUM(pow(r.severity, 2)/r.complexity)) as cvss_per_hour
        FROM results as r LEFT JOIN tmp_cves as c ON r.id = c.result_id
            LEFT JOIN tmp_solution_urls as s ON r.id = s.result_id
        WHERE r.processed = 0
            AND r.solution_type = "VendorFix"
            AND (LOWER(r.solution_text) LIKE "update %"
                OR LOWER(r.solution_text) LIKE "upgrade %")
        GROUP BY r.ip, r.vulnerable_service, r.port
        ORDER BY faculty, exploit_exists DESC, max_priority DESC;
        """
    )
    conn.commit()
    # mark results as processed
    conn.execute(
        """UPDATE results as r
        SET processed = 1
        WHERE EXISTS (
            SELECT *
            FROM major_updates as m
            WHERE r.ip = m.ip and r.vulnerable_service = m.service and r.port = m.port
        )
        """
    )
    conn.commit()
    # view minor_updates query for documentation
    conn.execute("DROP TABLE IF EXISTS high_priority_vulnerabilities")
    conn.execute(
        f"""CREATE TABLE high_priority_vulnerabilities AS
        WITH tmp_cves AS (
            SELECT r.id as result_id,
            rtrim(replace(group_concat(DISTINCT c.cve||'@!'), '@!,', x'0a'),'@!') as "cves"
            FROM results as r LEFT JOIN cves as c ON r.id = c.result_id
            WHERE r.processed = 0
            GROUP BY r.vulnerable_service, r.ip
        ),
        tmp_solution_urls AS (
            SELECT r.id as result_id,
            rtrim(replace(group_concat(DISTINCT s.url||'@!'), '@!,', x'0a'),'@!') as solutions
            FROM results as r LEFT JOIN solution_urls as s ON r.id = s.result_id
            WHERE r.processed = 0
            GROUP BY r.vulnerable_service, r.ip
```

```
656         )
657         SELECT
658             r.faculty as faculty,
659             r.ip as ip,
660             r.hostname as host_name,
661             r.vulnerable_service as service,
662             rtrim(replace(group_concat(DISTINCT r.name||'@!'), '@!,', x'0a'),'@!') as names,
663             r.port as port,
664             ROUND(MAX(r.priority), 2) as max_priority,
665             ROUND(SUM(r.priority), 2) as total_priority,
666             ROUND(AVG(r.priority), 2) as avg_priority,
667             COUNT(r.id) as occurences,
668             rtrim(replace(group_concat(DISTINCT r.solution_text||'@!'), '@!,', x'0a'),'@!')
669                 as solution_text,
670             MIN(r.vulnerable_version) as installed_version,
671             MAX(r.fixed_version) as fixed_version,
672             MAX(c.cves) as "cves",
673             MAX(s.solutions) as solutions,
674             MAX(r.exploit_exists) as exploit_exists,
675             MAX(r.complexity) as cost,
676                         r.severity as severity,
677                         sqrt(SUM(pow(r.severity, 2)/r.complexity)) as cvss_per_hour
678         FROM results as r LEFT JOIN tmp_cves as c ON r.id = c.result_id
679             LEFT JOIN tmp_solution_urls as s ON r.id = s.result_id
680         WHERE r.processed = 0
681         GROUP BY r.vulnerable_service, r.ip
682         HAVING (max_priority >= {HIGH_PRIORITY_THRESHOLD}
683             OR exploit_exists = 1) AND cost < 5.0
684         ORDER BY faculty, exploit_exists DESC, max_priority DESC
685         """
686     )
687     conn.commit()
688     # mark results as processed
689     conn.execute(
690         """UPDATE results as r
691             SET processed = 1
692             WHERE EXISTS (
693                 SELECT *
694                 FROM high_priority_vulnerabilities as v
695                 WHERE r.ip = v.ip AND r.vulnerable_service = v.service
696             );
697         """
698     )
699     conn.commit()
```

```
    # view minor_updates query for documentation
    conn.execute("DROP TABLE IF EXISTS wide_spread_vulnerabilities")
    conn.execute(
        f"""CREATE TABLE wide_spread_vulnerabilities AS
        WITH tmp_cves AS (
            SELECT r.id as result_id,
            rtrim(replace(group_concat(DISTINCT c.cve||'@!'), '@!,', x'0a'),'@!') as "cves"
            FROM results as r LEFT JOIN cves as c ON r.id = c.result_id
            WHERE r.processed = 0
            GROUP BY r.name, r.faculty
        ),
        tmp_solution_urls AS (
            SELECT r.id as result_id,
            rtrim(replace(group_concat(DISTINCT s.url||'@!'), '@!,', x'0a'),'@!') as solutions
            FROM results as r LEFT JOIN solution_urls as s ON r.id = s.result_id
            WHERE r.processed = 0
            GROUP BY r.name, r.faculty
        )
        SELECT
            r.faculty as faculty,
            rtrim(replace(group_concat(DISTINCT r.ip||'@!'), '@!,', x'0a'),'@!')
                as ip,
            rtrim(replace(group_concat(DISTINCT r.hostname||'@!'), '@!,', x'0a'),'@!')
                as host_name,
            r.vulnerable_service as service,
            r.name as name,
            rtrim(replace(group_concat(DISTINCT r.port||'@!'), '@!,', x'0a'),'@!')
                as port,
            ROUND(MAX(r.priority), 2) as max_priority,
            ROUND(SUM(r.priority), 2) as total_priority,
            ROUND(AVG(r.priority), 2) as avg_priority,
            COUNT(r.id) as occurences,
            rtrim(replace(group_concat(DISTINCT r.solution_text||'@!'), '@!,', x'0a'),'@!')
                as solution_text,
            MIN(r.vulnerable_version) as installed_version,
            MAX(r.fixed_version) as fixed_version,
            MAX(c.cves)  as "cves",
            MAX(s.solutions) as solutions,
            MAX(r.exploit_exists) as exploit_exists,
            AVG(r.complexity) as cost,
            r.severity as severity,
            sqrt(SUM(pow(r.severity, 2)/r.complexity)) as cvss_per_hour
        FROM results as r LEFT JOIN tmp_cves as c ON r.id = c.result_id
            LEFT JOIN tmp_solution_urls as s ON r.id = s.result_id
```

```
744         WHERE r.processed = 0
745         GROUP BY r.name, r.faculty
746         HAVING occurences > {WIDE_SPREAD_THRESHOLD}
747         ORDER BY faculty, exploit_exists DESC, occurences DESC, max_priority DESC, cost ASC
748         """
749     )
750     conn.commit()
751     # mark results as processed
752     conn.execute(
753         """UPDATE results as r
754             SET processed = 1
755             WHERE EXISTS (
756                 SELECT *
757                 FROM wide_spread_vulnerabilities as v
758                 WHERE r.faculty = v.faculty AND r.name = v.name
759             );
760         """
761     )
762     conn.commit()
763 
764     # view minor_updates query for documentation
765     conn.execute("DROP TABLE IF EXISTS high_effort_remediations")
766     conn.execute(
767         f"""CREATE TABLE high_effort_remediations AS
768         WITH tmp_cves AS (
769             SELECT r.id as result_id,
770             rtrim(replace(group_concat(DISTINCT c.cve||'@!'), '@!,', x'0a'),'@!') as "cves"
771             FROM results as r LEFT JOIN cves as c ON r.id = c.result_id
772             WHERE r.processed = 0
773             GROUP BY r.name, r.faculty
774         ),
775         tmp_solution_urls AS (
776             SELECT r.id as result_id,
777             rtrim(replace(group_concat(DISTINCT s.url||'@!'), '@!,', x'0a'),'@!') as solutions
778             FROM results as r LEFT JOIN solution_urls as s ON r.id = s.result_id
779             WHERE r.processed = 0
780             GROUP BY r.name, r.faculty
781         )
782         SELECT
783             r.faculty as faculty,
784             r.ip as ip,
785             r.hostname as host_name,
786             r.vulnerable_service as service,
787             r.name as name,
```

```
            r.port as port,
            ROUND(MAX(r.priority), 2) as max_priority,
            ROUND(SUM(r.priority), 2) as total_priority,
            ROUND(AVG(r.priority), 2) as avg_priority,
            COUNT(r.id) as occurences,
            rtrim(replace(group_concat(DISTINCT r.solution_text||'@!'), '@!,', x'0a'),'@!')
                as solution_text,
            MIN(r.vulnerable_version) as installed_version,
            MAX(r.fixed_version) as fixed_version,
            MAX(c.cves) as "cves",
            MAX(s.solutions) as solutions,
            MAX(r.exploit_exists) as exploit_exists,
            MAX(r.complexity) as cost,
            r.severity as severity,
            sqrt(SUM(pow(r.severity, 2)/r.complexity)) as cvss_per_hour
        FROM results as r LEFT JOIN tmp_cves as c ON r.id = c.result_id
            LEFT JOIN tmp_solution_urls as s ON r.id = s.result_id
        WHERE r.processed = 0
        GROUP BY r.name, r.faculty
        HAVING cost >= 5.0
        ORDER BY faculty, exploit_exists DESC, cost DESC, max_priority DESC
        """
    )
    conn.commit()

    conn.execute(
        """UPDATE results as r
            SET processed = 1
            WHERE EXISTS (
                SELECT *
                FROM high_effort_remediations as v
                WHERE r.ip = v.ip AND r.faculty = v.faculty AND r.name = v.name
        );
        """
    )
    conn.commit()
    # view minor_updates query for documentation
    conn.execute("DROP TABLE IF EXISTS remaining_vulnerabilities")
    conn.execute(
        f"""CREATE TABLE remaining_vulnerabilities AS
        WITH tmp_cves AS (
            SELECT r.id as result_id,
            rtrim(replace(group_concat(DISTINCT c.cve||'@!'), '@!,', x'0a'),'@!') as "cves"
            FROM results as r LEFT JOIN cves as c ON r.id = c.result_id
```

```
832             WHERE r.processed = 0
833         ),
834         tmp_solution_urls AS (
835             SELECT r.id as result_id,
836             rtrim(replace(group_concat(DISTINCT s.url||'@!'), '@!,', x'0a'),'@!') as solutions
837             FROM results as r LEFT JOIN solution_urls as s ON r.id = s.result_id
838             WHERE r.processed = 0
839         )
840         SELECT
841             r.faculty as faculty,
842             r.ip as ip,
843             r.hostname as host_name,
844             r.vulnerable_service as service,
845             r.name as name,
846             r.port as port,
847             r.priority as max_priority,
848             r.priority as total_priority,
849             r.priority as avg_priority,
850             1 as occurences,
851             r.solution_text as solution_text,
852             r.vulnerable_version as installed_version,
853             r.fixed_version as fixed_version,
854             c.cves as "cves",
855             s.solutions as solutions,
856             r.exploit_exists as exploit_exists,
857             r.complexity as cost,
858             r.severity as severity,
859             (r.severity/r.complexity) as cvss_per_hour
860         FROM results as r LEFT JOIN tmp_cves as c ON r.id = c.result_id
861             LEFT JOIN tmp_solution_urls as s ON r.id = s.result_id
862         WHERE r.processed = 0
863         ORDER BY faculty, exploit_exists DESC, max_priority DESC;
864         """
865     )
866     conn.commit()
867 
868     # the following query is used to generate stats about the results
869     conn.execute(
870         f"""CREATE TABLE IF NOT EXISTS stats (
871                 wide_spread_threshold,
872                 high_priority_threshold,
873 
874                 total_severity_before,
875                 avg_severity_before,
```

```
                total_priority_before,
                avg_priority_before,
                cost_before,
                total_vulns_before,
                high_severity_vulns_before,
                medium_severity_vulns_before,
                low_severity_vulns_before,
                cvss_per_hour_before,
                total_severity_after,
                avg_severity_after,
                total_priority_after,
                avg_priority_after,
                cost_after,
                total_vulns_after,
                high_severity_vulns_after,
                medium_severity_vulns_after,
                low_severity_vulns_after,
                cost_of_remediation,
                will_not_fix,
                none_available,
                workaround,
                mitigation,
                vendor_fix,
                total_vulns_to_remediate,
                cvss_per_hour,
                effectiveness
            )
        """
    )

    conn.execute(
        f""" INSERT INTO stats
    WITH totals_before AS (
        SELECT
            ROUND(SUM(severity), 2) as total_severity,
            ROUND(AVG(severity), 2) as avg_severity,
            ROUND(SUM(priority), 2) as total_priority,
            ROUND(AVG(priority), 2) as avg_priority,
            ROUND(SUM(complexity), 2) as total_cost,
            COUNT(*) as total_vulns,
            -- count of high severity vulns
            COUNT(IIF(severity >= 7.0, 1, NULL)) as high_severity_vulns,
            -- count of medium severity vulns
            COUNT(IIF(severity >= 4.0 AND severity < 7.0, 1, NULL))
```

```
920                 as medium_severity_vulns,
921             -- count of low severity vulns
922             COUNT(IIF(severity < 4.0, 1, NULL)) as low_severity_vulns,
923             -- RMSE of CVSS per hour
924             sqrt(SUM(pow(severity, 2)/complexity) / COUNT(*)) as cvss_per_hour
925         FROM results
926     ),
927     -- documenation for the following query can be found in the previous query
928     totals_after AS (
929         SELECT
930             ROUND(SUM(severity), 2) as total_severity,
931             ROUND(AVG(severity), 2) as avg_severity,
932             ROUND(SUM(priority), 2) as total_priority,
933             ROUND(AVG(priority), 2) as avg_priority,
934             ROUND(SUM(complexity), 2) as total_cost,
935             COUNT(*) as total_vulns,
936             COUNT(IIF(severity >= 7.0, 1, NULL)) as high_severity_vulns,
937             COUNT(IIF(severity >= 4.0 AND severity < 7.0, 1, NULL))
938                 as medium_severity_vulns,
939             COUNT(IIF(severity < 4.0, 1, NULL)) as low_severity_vulns
940         FROM results
941         WHERE processed = 0
942     ),
943     -- calculate the cost of remediation, CVSS per hour, and total vulns to remediate
944     total_remediation AS (
945         WITH tmp as (
946             SELECT
947                 MAX(cost) as total_cost,
948                 SUM(cost  = {WILL_NOT_FIX}) as will_not_fix,
949                 SUM(cost  = {NONE_AVAILABLE}) as none_available,
950                 SUM(cost  = {WORKAROUND}) as workaround,
951                 SUM(cost  = {MITIGATION}) as mitigation,
952                 SUM(cost  = {VENDOR_FIX}) as vendor_fix,
953                 COUNT(*) as total_vulns,
954                 SUM(cvss_per_hour) as cvss_per_hour
955             FROM minor_updates
956             UNION
957             SELECT
958                 MAX(cost) as total_cost,
959                 SUM(cost  = {WILL_NOT_FIX}) as will_not_fix,
960                 SUM(cost  = {NONE_AVAILABLE}) as none_available,
961                 SUM(cost  = {WORKAROUND}) as workaround,
962                 SUM(cost  = {MITIGATION}) as mitigation,
963                 SUM(cost  = {VENDOR_FIX}) as vendor_fix,
```

```
964                 COUNT(*) as total_vulns,
965                 SUM(cvss_per_hour) as cvss_per_hour
966             FROM major_updates
967             UNION
968             SELECT
969                 (MAX(cost) + (AVG(cost) * (SUM(occurences)-1)) * {REOCCURENCE_FACTOR})
970                     as total_cost,
971                 SUM(cost  = {WILL_NOT_FIX}) as will_not_fix,
972                 SUM(cost  = {NONE_AVAILABLE}) as none_available,
973                 SUM(cost  = {WORKAROUND}) as workaround,
974                 SUM(cost  = {MITIGATION}) as mitigation,
975                 SUM(cost  = {VENDOR_FIX}) as vendor_fix,
976                 COUNT(*) as total_vulns,
977                 SUM(cvss_per_hour) as cvss_per_hour
978             FROM high_priority_vulnerabilities
979             UNION
980             SELECT
981                 (MAX(cost) + (AVG(cost) * (SUM(occurences)-1)) * {REOCCURENCE_FACTOR})
982                     as total_cost,
983                 SUM(cost  = {WILL_NOT_FIX}) as will_not_fix,
984                 SUM(cost  = {NONE_AVAILABLE}) as none_available,
985                 SUM(cost  = {WORKAROUND}) as workaround,
986                 SUM(cost  = {MITIGATION}) as mitigation,
987                 SUM(cost  = {VENDOR_FIX}) as vendor_fix,
988                 COUNT(*) as total_vulns,
989                 SUM(cvss_per_hour) as cvss_per_hour
990             FROM wide_spread_vulnerabilities
991             UNION
992             SELECT
993                 (MAX(cost) + (AVG(cost) * (SUM(occurences)-1)) * {REOCCURENCE_FACTOR})
994                     as total_cost,
995                 SUM(cost  = {WILL_NOT_FIX}) as will_not_fix,
996                 SUM(cost  = {NONE_AVAILABLE}) as none_available,
997                 SUM(cost  = {WORKAROUND}) as workaround,
998                 SUM(cost  = {MITIGATION}) as mitigation,
999                 SUM(cost  = {VENDOR_FIX}) as vendor_fix,
1000                COUNT(*) as total_vulns,
1001                SUM(cvss_per_hour) as cvss_per_hour
1002            FROM high_effort_remediations
1003        )
1004        SELECT SUM(total_cost) as total_cost,
1005            SUM(will_not_fix) as will_not_fix,
1006            SUM(none_available) as none_available,
1007            SUM(workaround) as workaround,
```

```
1008            SUM(mitigation) as mitigation,
1009            SUM(vendor_fix) as vendor_fix,
1010            SUM(total_vulns) as total_vulns,
1011            SUM(cvss_per_hour) / SUM(total_vulns) as cvss_per_hour
1012        FROM tmp
1013    )
1014    SELECT
1015        {WIDE_SPREAD_THRESHOLD} as wide_spread_threshold,
1016        {HIGH_PRIORITY_THRESHOLD} as high_priority_threshold,
1017
1018        totals_before.total_severity as total_severity_before,
1019        totals_before.avg_severity as avg_severity_before,
1020        totals_before.total_priority as total_priority_before,
1021        totals_before.avg_priority as avg_priority_before,
1022        totals_before.total_cost as cost_before,
1023        totals_before.total_vulns as total_vulns_before,
1024        totals_before.high_severity_vulns as high_severity_vulns_before,
1025        totals_before.medium_severity_vulns as medium_severity_vulns_before,
1026        totals_before.low_severity_vulns as low_severity_vulns_before,
1027        totals_before.cvss_per_hour as cvss_per_hour_before,
1028        totals_after.total_severity as total_severity_after,
1029        totals_after.avg_severity as avg_severity_after,
1030        totals_after.total_priority as total_priority_after,
1031        totals_after.avg_priority as avg_priority_after,
1032        totals_after.total_cost as cost_after,
1033        totals_after.total_vulns as total_vulns_after,
1034        totals_after.high_severity_vulns as high_severity_vulns_after,
1035        totals_after.medium_severity_vulns as medium_severity_vulns_after,
1036        totals_after.low_severity_vulns as low_severity_vulns_after,
1037        ROUND(SUM(total_remediation.total_cost),2) as cost_of_remediation,
1038        ROUND(SUM(total_remediation.will_not_fix),2) as will_not_fix,
1039        ROUND(SUM(total_remediation.none_available),2) as none_available,
1040        ROUND(SUM(total_remediation.workaround),2) as workaround,
1041        ROUND(SUM(total_remediation.mitigation),2) as mitigation,
1042        ROUND(SUM(total_remediation.vendor_fix),2) as vendor_fix,
1043        ROUND(SUM(total_remediation.total_vulns),2) as total_vulns_to_remediate,
1044        ROUND(total_remediation.cvss_per_hour, 2) as cvss_per_hour,
1045        -- calculate effectiveness, prioritizing high severity vulns
1046        (
1047            3.0 * CAST(totals_before.high_severity_vulns as REAL)
1048            / (totals_before.high_severity_vulns - totals_after.high_severity_vulns)
1049            + 2.0 * CAST(totals_before.medium_severity_vulns as REAL)
1050            / (totals_before.medium_severity_vulns - totals_after.medium_severity_vulns)
1051            + CAST(totals_before.low_severity_vulns as REAL)
```

```
1052                / (totals_before.low_severity_vulns - totals_after.low_severity_vulns)
1053            ) as effectiveness
1054        FROM totals_before, totals_after, total_remediation
1055            """
1056        )
1057
1058        conn.commit()
1059
1060
1061    def close_db():
1062        conn.close()
1063
1064
1065    # get results from database
1066    # return Results object
1067    def get_results() -> Results:
1068        results = Results()
1069
1070        for row in conn.execute("SELECT * FROM faculties"):
1071            results.faculties.append(row[0])
1072
1073        for row in conn.execute("SELECT * FROM vuln_stats_by_faculty"):
1074            results.vuln_stats_by_faculty.append(row)
1075
1076        for row in conn.execute("SELECT * FROM major_updates"):
1077            results.major_updates.append(row)
1078
1079        for row in conn.execute("SELECT * FROM minor_updates"):
1080            results.minor_updates.append(row)
1081
1082        for row in conn.execute("SELECT * FROM high_priority_vulnerabilities"):
1083            results.high_priority_vulnerabilities.append(row)
1084
1085        for row in conn.execute("SELECT * FROM wide_spread_vulnerabilities"):
1086            results.wide_spread_vulnerabilities.append(row)
1087
1088        for row in conn.execute("SELECT * FROM high_effort_remediations"):
1089            results.high_effort_remediations.append(row)
1090
1091        for row in conn.execute("SELECT * FROM remaining_vulnerabilities"):
1092            results.remaining_vulnerabilities.append(row)
1093
1094        for row in conn.execute("SELECT * FROM stats"):
1095            results.stats.append(row)
```

```
1096
1097        return results
1098
1099
1100    # make a dictionary from a row of the database results
1101    # dictionary keys have to match the keys in the output templates
1102    def make_result_dict(row) -> dict:
1103        return {
1104            "host_ips": [line for line in row[1].split('\n') if line.strip()],
1105            "host_names": ['Unknown'] if row[2] is None else
1106            [line for line in row[2].split('\n') if line.strip()][:20],
1107            "host_service": row[3],
1108            "vuln_description": row[4],
1109            "port": [line for line in row[5].split('\n') if line.strip()],
1110            "max_priority": row[6],
1111            "total_priority": row[7],
1112            "avg_priority": row[8],
1113            "vuln_occurrences": row[9],
1114            "solution_texts": ['None'] if row[10] is None else
1115            [line for line in row[10].split('\n') if line.strip()],
1116            "oldest_installed_version": row[11],
1117            "latest_fixed_version": row[12] if row[12] != "99.99.99" and row[12] is not None
1118            else "EOL or Version Independent Problem, see Solutions for details",
1119            "cves": ['None'] if row[13] is None else row[13].splitlines()[:20],
1120            "solution_urls": ['None'] if row[14] is None else row[14].splitlines()[:20],
1121            "exploit_exists": row[15],
1122            "effort": row[16],
1123        }
1124
1125
1126    def export_results(results: Results):
1127        results_by_faculty = {}
1128        # make sure there is a key for each faculty
1129        faculties = results.faculties
1130        faculties = ['Unknown' if v is None else v for v in faculties]
1131        for faculty in faculties:
1132            results_by_faculty[faculty] = {}
1133
1134        # add vuln stats to each faculty
1135        for row in results.vuln_stats_by_faculty:
1136            faculty = 'Unknown' if row[0] is None else row[0]
1137            results_by_faculty[faculty]["vuln_stats"] = {}
1138            results_by_faculty[faculty]["vuln_stats"]["total_hosts"] = row[1]
1139            results_by_faculty[faculty]["vuln_stats"]["total_vulns"] = row[2]
```

```
1140            results_by_faculty[faculty]["vuln_stats"]["total_priority"] = row[3]
1141            results_by_faculty[faculty]["vuln_stats"]["total_severity"] = row[4]
1142            results_by_faculty[faculty]["vuln_stats"]["avg_vulns"] = row[5]
1143            results_by_faculty[faculty]["vuln_stats"]["avg_priority"] = row[6]
1144
1145        # add results to each faculty
1146        for row in results.major_updates:
1147            faculty = 'Unknown' if row[0] is None else row[0]
1148            if "major_updates" not in results_by_faculty[faculty]:
1149                results_by_faculty[faculty]["major_updates"] = []
1150            results_by_faculty[faculty]["major_updates"].append(
1151                make_result_dict(row)
1152            )
1153
1154        for row in results.minor_updates:
1155            faculty = 'Unknown' if row[0] is None else row[0]
1156            if "minor_updates" not in results_by_faculty[faculty]:
1157                results_by_faculty[faculty]["minor_updates"] = []
1158            results_by_faculty[faculty]["minor_updates"].append(
1159                make_result_dict(row)
1160            )
1161
1162        for row in results.high_priority_vulnerabilities:
1163            faculty = 'Unknown' if row[0] is None else row[0]
1164            if "high_priority_vulnerabilities" not in results_by_faculty[faculty]:
1165                results_by_faculty[faculty]["high_priority_vulnerabilities"] = []
1166            results_by_faculty[faculty]["high_priority_vulnerabilities"].append(
1167                make_result_dict(row)
1168            )
1169
1170        for row in results.wide_spread_vulnerabilities:
1171            faculty = 'Unknown' if row[0] is None else row[0]
1172            if "wide_spread_vulnerabilities" not in results_by_faculty[faculty]:
1173                results_by_faculty[faculty]["wide_spread_vulnerabilities"] = []
1174            results_by_faculty[faculty]["wide_spread_vulnerabilities"].append(
1175                make_result_dict(row)
1176            )
1177
1178        for row in results.high_effort_remediations:
1179            faculty = 'Unknown' if row[0] is None else row[0]
1180            if "high_effort_remediations" not in results_by_faculty[faculty]:
1181                results_by_faculty[faculty]["high_effort_remediations"] = []
1182            results_by_faculty[faculty]["high_effort_remediations"].append(
1183                make_result_dict(row)
```

```
1184        )
1185
1186    for row in results.remaining_vulnerabilities:
1187        faculty = 'Unknown' if row[0] is None else row[0]
1188        if "remaining_vulnerabilities" not in results_by_faculty[faculty]:
1189            results_by_faculty[faculty]["remaining_vulnerabilities"] = []
1190        results_by_faculty[faculty]["remaining_vulnerabilities"].append(
1191            make_result_dict(row)
1192        )
1193
1194    # determine the number of each vulnerability type per ip address
1195    for faculty in results_by_faculty:
1196        ips = {}
1197        for vuln_type in results_by_faculty[faculty]:
1198            # skip vuln_stats
1199            if vuln_type == "vuln_stats":
1200                continue
1201            for vuln in results_by_faculty[faculty][vuln_type]:
1202                for ip in vuln["host_ips"]:
1203                    if ip not in ips:
1204                        ips[ip] = {
1205                            "max_priority": vuln["max_priority"],
1206                            "exploit_exists": vuln["exploit_exists"],
1207                        }
1208                    if vuln_type not in ips[ip]:
1209                        ips[ip][vuln_type] = 0
1210                    ips[ip][vuln_type] += 1
1211                    if vuln["max_priority"] > ips[ip]["max_priority"]:
1212                        ips[ip]["max_priority"] = vuln["max_priority"]
1213        # sort ips by max priority
1214        results_by_faculty[faculty]["ips"] = sorted(
1215            ips.items(), key=lambda x: x[1]["max_priority"], reverse=True)
1216
1217    template_input = {
1218        "results": results_by_faculty,
1219        "faculties": faculties,
1220    }
1221
1222    # load template for all faculties
1223    templateLoader = jinja2.FileSystemLoader(searchpath="./")
1224    templateEnv = jinja2.Environment(loader=templateLoader)
1225    template = templateEnv.get_template(TEMPLATEFILE)
1226    # render template
1227    outputText = template.render(items=template_input)
```

```
    outfile = join(OUTPUTDIR, f"{OUTPUTFILE}.html")
    with open(outfile, "w") as f:
        f.write(outputText)

    # send email
    email = get_email(faculty)
    if email is not None:
        send_mail(outfile, email, faculty)

    # load template for each faculty
    for faculty in faculties:
        if faculty == 'Unknown':
            continue
        template = templateEnv.get_template(f"single_faculty_{TEMPLATEFILE}")
        outputText = template.render(
            faculty=faculty, results=results_by_faculty[faculty])
        outfile = join(
            OUTPUTDIR, f"""{OUTPUTFILE}-{faculty.replace("/", "_")}.html""")
        with open(outfile, "w") as f:
            f.write(outputText)
        # send email
        email = get_email(faculty)
        if email is not None:
            send_mail(outfile, email, faculty)


if __name__ == "__main__":

    args = sys.argv

    if len(args) < 2:
        print("\nRunning full prioritization\n")
        print("\nConnecting to database...\n")
        connect_to_database()
        print("\nInitializing database...\n")
        init_database()
        print("\nPre Processing... (this may take a while)\n")
        pre_process()
        print("\nAnalyzing results...\n")
        analyze()
        results = get_results()
        print("\nExporting results...\n")
        export_results(results)
        close_db()
```

```
1272            exit(0)
1273
1274        arg = args[1]
1275
1276        if arg == "help":
1277            print(
1278                "\nUsage: python3 prioritize.py [full|init|update|test|analyze]\n")
1279            exit(0)
1280
1281        if arg in ["full", "init ", "update", "test", "analyze", "threshold_efficiency"]:
1282            print("\nConnecting to database...\n")
1283            connect_to_database()
1284
1285        if arg in ["full", "init", "test"]:
1286            print("\nInitializing database...\n")
1287            init_database()
1288
1289        if arg in ["full", "update", "test"]:
1290            print("\nPre Processing... (this may take a while)\n")
1291            pre_process()
1292
1293        if arg in ["full", "analyze", "update", "test"]:
1294            print("\nAnalyzing results...\n")
1295            analyze()
1296
1297        if arg == "threshold_efficiency":
1298            for i in range(1, 19, 2):
1299                for k in range(10, 41, 5):
1300                    HIGH_PRIORITY_THRESHOLD = i
1301                    WIDE_SPREAD_THRESHOLD = k
1302                analyze()
1303
1304        if arg in ["threshold_efficiency",]:
1305            results = get_results()
1306            for result in results.stats:
1307                print(f"{result}")
1308
1309        if arg in ["full", "analyze", "update", "test"]:
1310            results = get_results()
1311            print("\nExporting results...\n")
1312            export_results(results)
1313
1314        if arg in ["full", "init", "update", "test", "analyze", "threshold_efficiency"]:
1315            close_db()
```

```

    exit(0)
```

B.6. Script: send_mail.py

```python
import smtplib
import ssl

from email import encoders
from email.mime.base import MIMEBase
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

from get_envs import get_envs

env = get_envs()


def send_mail(file, email_address, faculty):
    message = MIMEMultipart()
    message["From"] = env['EMAIL_ADDR']
    message["To"] = email_address
    message["Subject"] = f"Vulnerability Report für die Einrichtung: {faculty}"

    # Add body to email
    message.attach(
        MIMEText(f"{open('mail_body.txt', 'r').read()}\r\n", "plain"))

    filename = file  # In same directory as script

    # Read attachement
    with open(filename, "rb") as attachment:
        part = MIMEBase("application", "octet-stream")
        part.set_payload(attachment.read())
    encoders.encode_base64(part)

    # Set filename for attachement
    part.add_header(
        "Content-Disposition",
        f"attachment; filename= {filename}",
    )

    # Add attachement to message
    message.attach(part)
```

```python

    # Convert message to string
    text = message.as_string()

    # Send mail over SMTP
    context = ssl.create_default_context()
    with smtplib.SMTP(env['EMAIL_HOST'], env['EMAIL_PORT']) as server:
        server.ehlo()
        server.starttls(context=context)
        server.ehlo()
        server.login(env['EMAIL_ADDR'], env['PASSWORD'])
        server.sendmail(env['EMAIL_ADDR'], email_address, text)
```

B.7. Script: get_envs.py

```python
def get_envs():
    result_dict = {}
    with open(".env", 'r') as file:
        for line in file:
            line = line.strip()
            if line:
                key, value = line.split('=')
                result_dict[key.strip()] = value.strip()
    return result_dict
```

References

[Lin04] Pete Lindstrom. *Network vs. Host-Based Vulnerability Management*. Malvern, PA, USA, 2004.

[Sca+08] Karen Scarfone et al. *Technical Guide to Information Security Testing and Assessment*. National Institute of Standards and Technology, 2008.

[Hol+11] H. Holm et al. "A quantitative evaluation of vulnerability scanning". In: *Information Management & Computer Security, Vol. 19 No. 4*. 2011, pp. 231–247. DOI: 10.1108/09685221111173058.

[Sch+11] Klaus Schwab et al. *Personal Data: The Emergence of a New Asset Class*. Tech. rep. Cologny/Geneva, Switzerland: World Economic Forum, 2011.

[Kro13] Hans Georg Krojanski. *Physische IT Sicherheit*. Sicherheitstage WS 2012/13. 2013.

[MK15] Yuma Makino and Vitaly Klyuev. "Evaluation of web vulnerability scanners". In: *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*. Vol. 1. 2015, pp. 399–402. DOI: 10.1109/IDAACS.2015.7340766.

[MGS15] Yisroel Mirsky, Noam Gross, and Asaf Shabtai. "Up-High to Down-Low: Applying Machine Learning to an Exploit Database". In: *Innovative Security Solutions for Information Technology and Communications*. Ed. by Ion Bica, David Naccache, and Emil Simion. Cham: Springer International Publishing, 2015, pp. 184–200. ISBN: 978-3-319-27179-8.

[Alm+17] Mohammed Almukaynizi et al. "Predicting Cyber Threats through Hacker Social Networks in Darkweb and Deepweb Forums". In: *Proceedings of the 2017 International Conference of The Computational Social Science Society of the Americas*. CSS 2017. Santa Fe, NM, USA: Association for Computing Machinery, 2017. ISBN: 9781450352697. DOI: 10.1145/3145574.3145590. URL: https://doi.org/10.1145/3145574.3145590.

[Har+18] Christopher R. Harrell et al. "Vulnerability Assessment, Remediation, and Automated Reporting: Case Studies of Higher Education Institutions". In: *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*. 2018, pp. 148–153. DOI: 10.1109/ISI.2018.8587380.

[Alp+19] Kenneth Alperin et al. "Risk Prioritization by Leveraging Latent Vulnerability Features in a Contested Environment". In: AISec'19. London, United Kingdom: Association for Computing Machinery, 2019, pp. 49–57. ISBN: 9781450368339. DOI: 10.1145/3338501.3357365. URL: https://doi.org/10.1145/3338501.3357365.

[Bad20] Land Baden-Württemberg. "Hochschulfinanzierungsvereinbarung Baden-Württemberg 2021–2025". In: *Vereinbarung des Landes Baden-Württemberg mit den Hochschulen des Landes Baden-Württemberg vom 31. März 202*. Land Baden-Württemberg, 2020.

[CDN20] Agustín Chancusi, Paúl Diestra, and Damián Nicolalde. "Vulnerability analysis of the exposed public IPs in a higher education institution". In: *2020 the 10th International Conference on Communication and Network Security*. 2020, pp. 83–90.

[Ger+20] Cornelia Gerdenitsch et al. "Work gamification: Effects on enjoyment, productivity and the role of leadership". In: *Electronic Commerce Research and Applications* 43 (2020), p. 100994. ISSN: 1567-4223. DOI: https://doi.org/10.1016/j.elerap.2020.100994. URL: https://www.sciencedirect.com/science/article/pii/S1567422320300715.

[OWA20] OWASP. *Vulnerability Scanning Tools*. 2020. URL: https://owasp.org/www-community/Vulnerability_Scanning_Tools (visited on 06/01/2023).

[CIS21] CISA. *Reducing the Significant Risk of Known Exploited Vulnerabilities*. Cybersecurity & Infrastructure Security Agency, 2021.

[Spr+21] Jonathan Spring et al. "Time to Change the CVSS?" In: *IEEE Security & Privacy* 19.2 (2021), pp. 74–78. DOI: 10.1109/MSEC.2020.3044475.

[Gre22a] Greenbone. *Reports and Vulnerability Management*. 2022. URL: https://docs.greenbone.net/GCS-Manual/gcs/en/reports.html (visited on 04/24/2023).

[Gre22b] Greenbone. *Reports and Vulnerability Management*. 2022. URL: https://docs.greenbone.net/GSM-Manual/gos-21.04/en/scanning.html (visited on 05/20/2023).

[Lat22] Lyudmil Latinov. *MD5, SHA-1, SHA-256 and SHA-512 speed performance*. 2022. URL: https://automationrhapsody.com/md5-sha-1-sha-256-sha-512-speed-performance/ (visited on 06/29/2023).

[Rey+22] Jorge Reyes et al. "An Environment-Specific Prioritization Model for Information-Security Vulnerabilities Based on Risk Factor Analysis". In: *Electronics* 11.9 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11091334. URL: https://www.mdpi.com/2079-9292/11/9/1334.

[Sha+22] Ankit Shah et al. "Vulnerability Selection for Remediation: An Empirical Analysis". In: *The Journal of Defense Modeling and Simulation* 19.1 (2022), pp. 13–22. DOI: 10.1177/1548512919874129. URL: https://doi.org/10.1177/1548512919874129.

[Wag22] Jan-Oliver Wagner. *Active and Passive Vulnerability Scans – One Step Ahead of Cyber Criminals*. 2022. URL: https://www.greenbone.net/en/blog/active-passive-scans/ (visited on 06/01/2023).

[Jac+23] Jay Jacobs et al. *Enhancing Vulnerability Prioritization: Data-Driven Exploit Predictions with Community-Driven Insights*. 2023. arXiv: 2302.14172 [cs.CR].

[Krs23] Stefan Krstevski. *Warum Universitäten und Forschungsinstitute beliebte Ziele für Hacker sind*. 2023. URL: https://www.dataguard.de/blog/beliebt-bei-hackern-wie-sich-unis-forschungsinstitute-schuetzen-sollten (visited on 04/24/2023).

[Wal23] Chris Wallis. *Vulnerability Scanning Frequency Best Practices*. 2023. URL: https://www.intruder.io/blog/vulnerability-scanning-frequency-best-practices (visited on 06/27/2023).

[Connd] SQLite Consortium. *About SQLite*. n.d. URL: https://www.sqlite.org/about.html (visited on 06/26/2023).

[Ecknd] Oliver Eck. *DBSYS2 – 3. Analytische Datenbanken*. n.d.

[Grend] Greenbone. *OpenVAS - Open Vulnerability Assessment Scanner*. n.d. URL: https://www.openvas.org (visited on 04/17/2023).

[Konnd] Bert Kondruss. *Cyberangriffe auf Universitäten*. n.d. URL: https://konbriefing.com/de-topics/cyber-angriffe-universitaeten.html (visited on 04/24/2023).

[Lyond] Gordon Lyon. *Nmap: Discover your network*. n.d. URL: https://nmap.org (visited on 04/17/2023).

[Micnd] Microsoft. *Microsoft Security Intelligence*. n.d. URL: https://www.microsoft.com/en-us/wdsi/threats (visited on 04/24/2023).

[NISnd] NIST. *NVD - Vulnerability Metrics*. n.d. URL: https://nvd.nist.gov/vuln-metrics/cvss (visited on 06/26/2023).

[Offnd] OffSec. *About The Exploit Database*. n.d. URL: https://www.exploit-db.com/about-exploit-db (visited on 06/27/2023).

[Prend] Tom Preston-Werner. *Semantic Versioning 2.0.0*. n.d. URL: https://semver.org (visited on 06/18/2023).