UNIVERSITÀ DI PISA

Information Engineering Department

Master degree in Computer Engineering,
cursus Cybersecurity

A CPE matcher for personalized CVE filtering

Supervisors:

Prof. Giuseppe Lettieri

Prof. Gianluca Dini

Dott. Walter Volpi

Candidate:

Guillaume Quint

Academic Year 2023/2024

A nonno Ezio

Abstract

The identification and matching of Common Platform Enumerations (CPEs) play a critical role in accurately filtering the vulnerabilities described by Common Vulnerabilities and Exposures (CVEs). This study evaluates various matching techniques, focusing on their applicability to structured data such as CPE fields. It investigates approaches to enhance match accuracy by exploiting the structured nature of CPEs and incorporating additional metadata to deal with unstructured and noisy inputs. The research also addresses challenges in balancing accuracy and usability, proposing methods to optimize the trade-offs between automation and manual intervention. These findings aim to inform the design and implementation of an effective and adaptable algorithm for personalized CVE filtering.

Contents

List of Figures

Chapter 1

Introduction

1.1 The importance of having a detailed software inventory

The world's cybersecurity landscape has seen a dramatic transformation in recent years, and supply chain attacks have emerged as one of the most significant threats to organizational security.

These sophisticated attacks exploit the complex interconnections within modern digital infrastructure, targeting vulnerabilities in third-party vendors, software dependencies, and service providers to compromise primary targets through trusted channels. The scale and frequency of supply chain attacks have increased at an alarming rate. The 2023 Data Breach report by ITRC [14] reveals that organizations impacted by such attacks have increased by more than 2,600% since 2018, with affected entities rising by 15% in 2023 alone, affecting over 54 million victims.

Juniper Research estimates that, without a paradigm shift in software supply chain cybersecurity management, cyberattacks targeting software supply chains will cost the world economy an estimated $80.6 billion in lost revenue and damages annually by 2026 [17].

Recent real-world events

Several high-profile incidents have highlighted the devastating potential of supply chain attacks. The 2020 SolarWinds breach represented a critical moment in supply chain security, where compromised software updates affected over 18,000 organizations, demonstrating the cascading impact of such attacks. More recent incidents, such as the 2023 3CX Phone System compromise, have further emphasized vulnerabilities in trusted software distribution channels. The discovery of a malicious backdoor in XZ Utils (versions 5.6.0 and 5.6.1) in March 2024 represented another critical incident, affecting numerous Linux distributions and allowing unauthorized command execution on compromised systems.

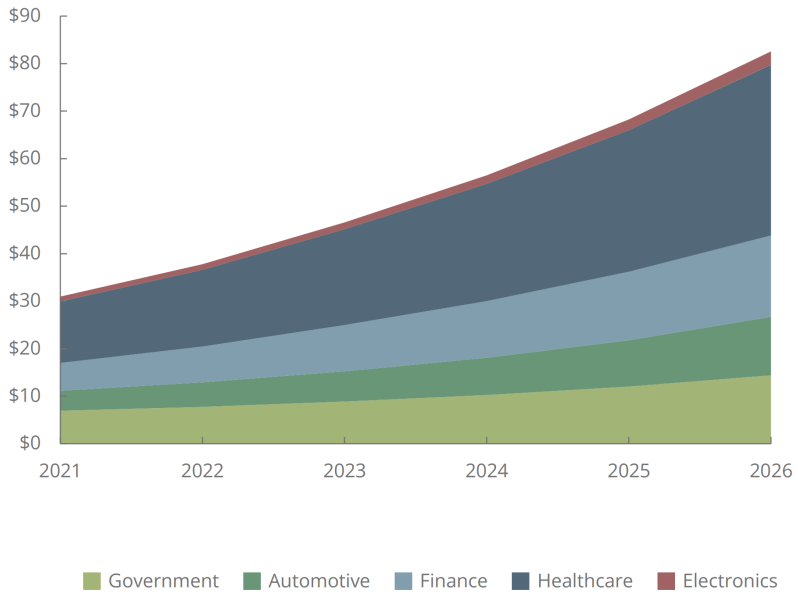


Figure 1.1: Revenue Losses Attributable to Supply Chain Cyber attacks ($billion), Split by Sector, 2021-2026, as projected by Juniper Research in 2023

Open source prevalence in all major economic sectors

The increasing prevalence of these attacks can be attributed to several structural vulnerabilities in modern digital infrastructure. Open-source software components, while essential for contemporary software development, have become a significant vector for supply chain attacks, with research[7] [15] [21] indicating that 64% of affected companies attribute their compromises to open-source dependencies.

Supply chain attacks have increased in intensity within all major strategic sectors, including Automotive, Consumer Electronic Devices, Finance, Government, Healthcare and Smart Cities. Revenue losses, split by sector, have been estimated by Juniper Research [17] and are reported in Figure 1.1

Industry analysts, including Gartner , project a concerning trajectory for supply chain security. Their research suggests that by 2025, approximately 45% of organizations worldwide will have experienced attacks on their software supply chains, representing a three-fold increase from 2021 levels. This projection underscores the urgent need for improved security measures and comprehensive supply chain risk management strategies.

1.2 Risk management through software identification

The European Union has significantly strengthened its cybersecurity framework with the introduction of the NIS2 Directive (Directive (EU) 2022/2555)[1], which came into

[1]https://eur-lex.europa.eu/eli/dir/2022/2555

force on January 16, 2023, replacing the original NIS Directive. This updated directive aims to establish a high common level of cybersecurity across member states by setting stricter security requirements for a broader range of essential and important entities. A critical aspect of NIS2 is the emphasis on comprehensive risk management measures, which include the need for robust asset management.This entails maintaining an up-to-date inventory of all network and information systems, both hardware and software components. Such detailed inventories are vital for identifying vulnerabilities, managing risks, and ensuring timely responses to incidents. Given the complexity and scale of modern IT environments, automating software identification becomes essential. Automation facilitates the continuous monitoring and cataloging of software assets, ensuring that organizations can maintain accurate software inventories, identify vulnerabilities quickly, and ensure compliance with NIS2 requirements.

Chapter 2

CPE Matching an asset inventory

By CPE Matching we refer to the challenge of accurately identifying and correlating software, hardware, or operating system products of an asset inventory using standardized CPE (Common Platform Enumeration) names.

It should not be confused with the CPE 2.3 Name Matching Specification, which is talked about in 2.2.2

2.1 The naming problem

Different vendors or tools use inconsistent naming conventions for the same product. Software versions are represented differently (e.g., "10.0" vs. "10.0.0"). There is a high variability in how wildcards and partial matches are handled. Also, it is difficult to match vendor and product names due to spelling variations (e.g., "Microsoft Corporation" vs. "Microsoft" vs "MS"), and there are ambiguities in distinguishing different product editions or platforms.

2.2 CPE

Common Platform Enumeration (CPE) is a standardized method of describing and identifying classes of applications, operating systems, and hardware devices present among an enterprise's computing assets. CPE does not identify unique instantiations of products on systems, such as the installation of XYZ Visualizer Enterprise Suite 4.2.3 with serial number Q472B987P113. Rather, CPE identifies abstract classes of products, such as XYZ Visualizer Enterprise Suite 4.2.3, XYZ Visualizer Enterprise Suite (all versions), or XYZ Visualizer (all variations).

IT management tools can collect information about installed products, identifying these products using their CPE names, and then use this standardized information to help make fully or partially automated decisions regarding the assets. For ex-

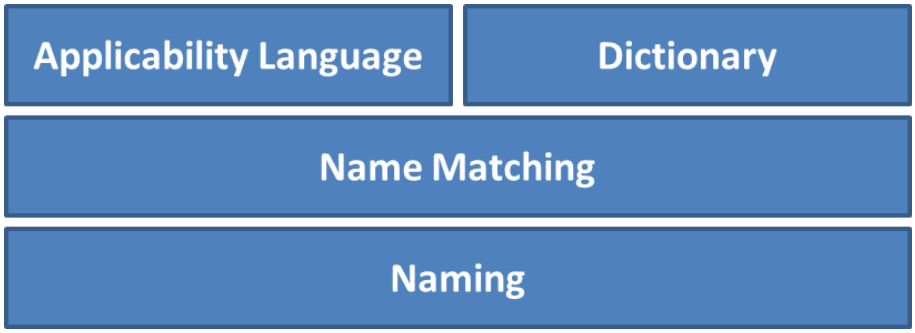


Figure 2.1: CPE stack

ample, identifying the presence of XYZ Visualizer Enterprise Suite could trigger a vulnerability management tool to check the system for known vulnerabilities in the software, and also trigger a configuration management tool to verify that the software is configured securely in accordance with the organization's policies.

The current version of CPE is 2.3. CPE 2.3 is defined through a set of specifications in a stack-based model, where capabilities are based on simpler, more narrowly defined elements that are specified lower in the stack. This design opens opportunities for innovation, as novel capabilities can be defined by combining only the needed elements, and the impacts of change can be better compartmentalized and managed.

Figure 2.1 shows the current CPE 2.3 stack, with the most fundamental layer (Naming) at the bottom. Each higher layer builds on top of the layers below it.

The CPE format is based upon the generic syntax for Uniform Resource Identifiers (URI). The CPE Product Dictionary provides an agreed upon list of official CPE names. The dictionary is provided in XML format and is available to the general public, but is soon to be deprecated[1]. The new 2.0 API allows RESTful requests to access the dictionary in a JSON format. The CPE Dictionary is hosted and maintained at NIST, and may be used by nongovernmental organizations on a voluntary basis.

The 2.3 version of CPE has the following format:

```
cpe:<cpe_version>:<part>:<vendor>:<product>:<version>:<update>:
<edition>:<language>:<sw_edition>:<target_sw>:<target_hw>:<other>
```

CPE allows the use of wildcards to perform grouping expressions, for example:

```
cpe:2.3:a:ntp:ntp:4.2.8:p3:*:*:*:*:*:*
cpe:2.3:o:microsoft:windows_7:-:sp2:*:*:*:*:*:*
cpe:2.3:a:microsoft:internet_explorer:8.0.6001:beta:*:*:*:*:*:*
```

Where a '-' indicates the absence of information and '*' represents any value for that field.

A brief overview of the four CPE v2.3 specifications are included below

[1]https://nvd.nist.gov/General/News/change-timeline

2.2.1 Naming

The CPE 2.3 Naming Specification[9] defines standardized methods for assigning names to IT product classes. An example is the following name representing Microsoft Internet Explorer 8.0.6001 Beta:

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

This method of naming is known as a well-formed CPE name (WFN). It is an abstract logical construction. The CPE Naming Specification defines procedures for binding WFNs to machine-readable encodings (a CPE URI), as well as unbinding those encodings back to WFNs.

We should be particularly careful on what this specification is not about. Section 5 enumerates a list of aspects that are considered outside the scope of the CPE Naming specification. There, we also find:

> "Other aspects of product naming and description that are outside the scope of the CPE Naming specification are: [...]
>
> - Defining procedures and guidelines for assigning "correct" or "valid" values to attributes of product descriptions or identifiers"

It is clear that the specification cannot specify the exact content of each WFN's field for each software, but instead defines what it should conceptually contain. For example, here is how values for the Product field are defined (Section 5.3.3.3 of the reference document):

> "Values for this attribute SHOULD describe or identify the most common and recognizable title or name of the product. Values for this attribute SHOULD be selected from an attribute-specific valid-values list, which MAY be defined by other specifications that utilize this specification. Any character string meeting the requirements for WFNs (cf. 5.3.2) MAY be specified as the value of the attribute."

This still leaves a very high margin of interpretation on what constitutes the best identifier for each field.

2.2.2 Name Matching

The CPE 2.3 Name Matching Specification[22] defines a method for conducting a one-to-one comparison of a source CPE name to a target CPE name. It is not about matching a software name to a CPE name. By logically comparing CPE names as sets of values, CPE Name Matching methods can determine if common set relations hold. For example, CPE Name Matching can determine if the source and target names are equal, if one of the names is a subset of the other, or if the names are disjoint.

One example of the value of CPE Name Matching is in determining if a particular product is installed on a system. Suppose that an organization is identifying which of its systems have any variation of Microsoft Internet Explorer 8 installed. This could be represented with the following well-formed CPE name (WFN):

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.*",update=ANY,edition=ANY,language=ANY]
```

An asset management tool could collect information on the software installed on a system and compare its Internet Explorer installation characteristics to the WFN above. Suppose that the WFN for a particular installed instance of Internet Explorer was reported as:

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.0\.6001",update=NA,edition=NA,language="en\-us"]
```

Using these two example WFNs, CPE Name Matching methods perform a pairwise comparison of attribute values in the first (source) WFN to those in the second (target) WFN, yielding a list of the set relations between each pair of attributes (e.g., equal, superset). This list of comparison results is then assessed, leading to a determination that the first WFN represents a superset of the second WFN. This can be interpreted to mean that the system being examined does indeed have a variation of Microsoft Internet Explorer 8 installed.

Another attribute comparison example, which uses a different software product, is shown in Table 2.1 and displays the relation set obtained for each pair of attributes.

Attribute	**Part**	**Vendor**	**Product**	**Version**	**Update**	**Edition**
Source Value	a	Adobe	ANY	9.*	ANY	PalmOS
Target Value	a	ANY	Reader	9.3.2	NA	NA
Relation Set	$=$	$\subset$	$\supset$	$\supset$	$\supset$	$\neq$

Table 2.1: Attribute Comparison Example

After obtaining the list of attribute relations, we follow the schema presented in Table 2.2

If Attribute Relation Set =	**Then Name Comparison Relation**
If any attribute relation is DISJOINT ($\neq$)	Then CPE name relation is DISJOINT ($\neq$)
If all attribute relations are EQUAL ($=$)	Then CPE name relation is EQUAL ($=$)
If all attribute relations are SUBSET ($\subset$)	Then CPE name relation is SUBSET ($\subset$)
If all attribute relations are SUPERSET ($\supset$) or EQUAL ($=$)	Then CPE name relation is SUPERSET ($\supset$)

Table 2.2: Required CPE Name Comparison Relations

In this example, we can assess that the source and target CPEs are DISJOINT. Given the ambiguities derived by the Naming specification, NIST cannot define a rigorous way to perform name matching across all existing software classes and

instead leaves it as an implementation detail. In practice, a simple string comparison is not sufficient to assess if two names correspond to the same value because what is actually important is the semantic relation between two names. Taken from section 5 of the official specification document:

> "CPE's developers have chosen not to define a single notion of "name match" because experience has shown that name matching distinctions are often use-case dependent. For example, when a source WFN is generated from the sparse results of a non-authenticated asset inventory tool, matching of only one or two CPE attribute values may constitute a meaningful "name match" for some applications. In contrast, when both the source and target WFNs are fully specified, common names in an authoritative CPE dictionary, it may be reasonable to decide that a "name match" requires an exact match of all CPE attribute values in both names. In order to remain flexible enough to support these and other use cases, the CPE Name Matching specification leaves the majority of decisions about what constitutes a "name match" to CPE implementers at design time"

2.2.3 Dictionary

The CPE 2.3 Dictionary Specification[10] defines a standardized method for creating and managing CPE dictionaries. A dictionary is a repository of CPE names and metadata associated with the names. Each CPE name in the dictionary identifies a single class of IT product in the world. The word "class" here signifies that the object identified is not a physical instantiation of a product on a system, but rather the abstract model of that product. Although organizations may use a CPE name to represent either a single product class or a set of multiple product classes, a CPE dictionary stores only bound forms of well-formed CPE names (WFNs) that identify a single product class, not a set of product classes. These single product-class WFNs in bound form are referred to as identifier names.

NIST hosts the Official CPE Dictionary, which is the authoritative repository of identifier names.

2.2.4 Applicability Language

The CPE 2.3 Applicability Language Specification[32] defines a standardized way to describe IT platforms by forming complex logical expressions out of individual CPE names and references to checks. For example, one could use the CPE 2.3 Applicability Language to combine the CPE name for an operating system (such as Microsoft Windows XP), the CPE name for an application running on that operating system (such as Microsoft Office 2007), and a reference to a check for a particular value of a certain configuration setting (such as the wireless network card being enabled in the operating system). These logical expressions are called applicability statements, because they are used to designate which platforms particular guidance,

policies, etc. apply to. Applicability statements can be used by tools to determine whether a target system is an instance of a particular platform.

2.3 Matcher requirements

The increasing prevalence of complex software across economic sectors has led modern Security Operations Centers (SOCs) often needing to manage multiple clients simultaneously. Each client's asset inventory may range from a few workstations to thousands of virtual machines, resulting in a substantial volume of software installations requiring analysis.

Furthermore, the Common Platform Enumeration (CPE) database is continuously evolving. New software products and versions are released daily, while vendor names frequently change due to acquisitions or rebranding. Consequently, maintaining an up-to-date CPE representation of software inventories is essential. To ensure accuracy, the entire software stack should be analyzed before every update, which occurs every 24 hours.

A robust matching solution must efficiently process large software inventories while accommodating heterogeneous data sources. Different clients may utilize distinct inventory tools, often employing various file formats. Thus, the matching approach should be agnostic to data formats and capable of handling noisy input data through appropriate data-cleaning techniques.

Given the complexity and scale of CPE matching, automation is critical. Manual matching is time-consuming, requires specialized expertise, and diverts resources from other cybersecurity tasks. While fully automated solutions remain imperfect, a hybrid approach where automated matching is supplemented by human verification can significantly enhance efficiency.

Moreover, the system should provide transparency in its decision-making process, allowing users to understand why a particular match was made. Designing such a mechanism requires acknowledging the probability of errors and equipping users with feedback and tuning options to refine results as needed.

This study takes into consideration the experience of consorzio Metis, which embodies the SOC and CSIRT role for multiple entities, including *Regione Toscana*, *Servizio Sanitario Regionale* and *Azienda Ospedaliero-Universitaria Meyer*.

Chapter 3

State of the art

We present a brief overview of the most relevant works in the field of software inventory management and vulnerability assessment.

3.1 SCAP

The Security Content Automation Protocol (SCAP) by NIST is a method for using specific standards to enable automated vulnerability management, measurement, and policy compliance evaluation of systems deployed in an organization, including e.g., FISMA (Federal Information Security Management Act, 2002) compliance. The National Vulnerability Database (NVD) is the U.S. government content repository for SCAP. An example of a SCAP implementation is OpenSCAP https://www.open-scap.org/.

SCAP is a suite of tools that have been compiled to be compatible with various protocols for things like configuration management, compliance requirements, software flaws, or vulnerabilities patching. Accumulation of these standards provides a means for data to be communicated between humans and machines efficiently. The objective of the framework is to promote a communal approach to the implementation of automated security mechanisms that are not monopolized

SCAP defines how the CVE (Common Vulnerabilities and Exposures) and CPE (Common Platform Enumeration) standards (referred to as SCAP 'Components') are combined together with other components, such as CVSS (Common Vulnerability Scoring System), CCE (Common Configuration Enumeration), OVAL (Open Vulnerability and Assessment Language) and more.

The main disadvantage linked with implementing SCAP comes from the need for SOCs to provide an additional asset inventory management service, because OpenSCAP relies on its internal asset inventory scanner. This is often unfeasible in practice, for responsibility reasons, as it is much more difficult to maintain control over devices that are primarily used outside the SOC's perimeter. This solution is preferable only in case a SOC has complete control over a single client's inventory. For our wanted use-case, a more open approach is needed, which does not require a

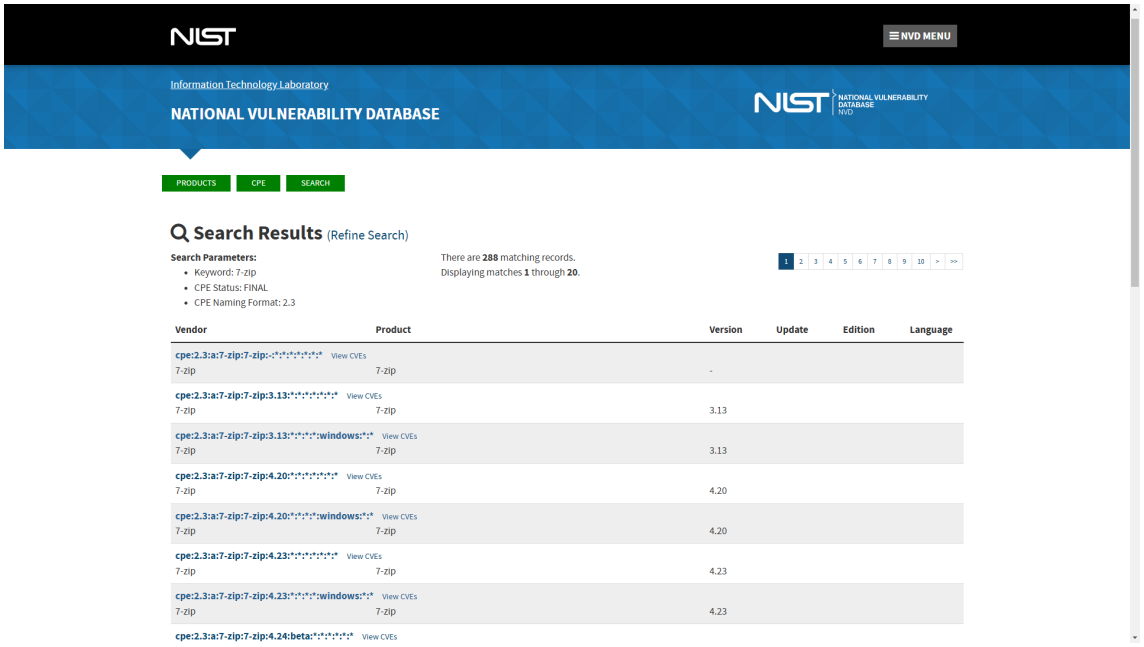


Figure 3.1: Too many records get returned with an under-specified query ('7-zip')

managed asset inventory and instead is capable of adapting to any inventory censing tool.

3.2 NIST's NVD search for CPEs

https://nvd.nist.gov/products/cpe/search The CPE Name search retrieves exact matches and records containing the components specified in the user-provided CPE Name.

A key limitation of this approach is its sensitivity to query specificity. Overly specific queries may yield no results, while underspecified queries can return an overwhelming number of candidates. Additionally, the reliance on a web service introduces latency, which may hinder real-time analysis and responsiveness.

To better demonstrate the retrieval behavior, we'll take as an example the voluntarily ambiguous software sample `7-zip 9.20`, which is known to be present inside the database. This should not be taken as an extensive benchmark; instead is only used for demonstrating how the underlying algorithm performs.

If we only specify the name part, we obtain 288 matching records as shown in Figure 3.1: too many to be considered useful.

Only once we add the version information does the result set approaches a more manageable size of only 6 records, as shown in Figure 3.2

Although we can observe that a minor inconvenience arises from the ranking choice. This is debatable, but in this specific instance, a more precise CPE like `cpe:2.3:a:7-zip:7-zip:9.20:*:*:*:*:windows:*:*` gets ranked higher than a broader one like `cpe:2.3:a:7-zip:p7zip:9.20:*:*:*:*:*:*:*` even though no

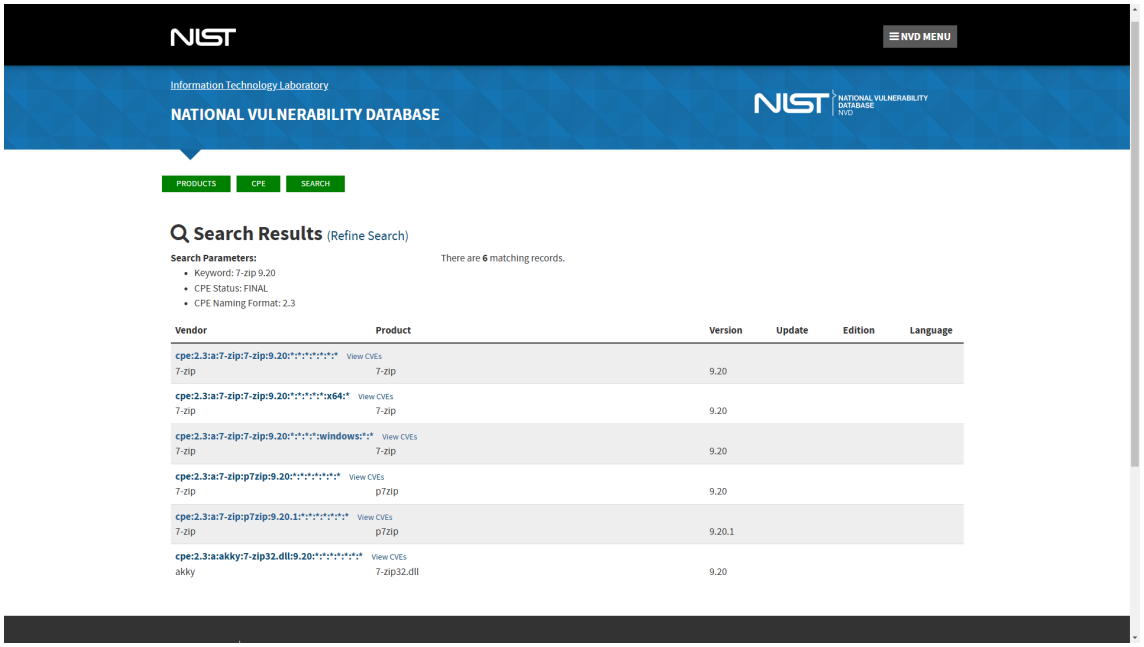


Figure 3.2: Only 6 records get returned if the query is extremely well polished ('7-zip 9.20')

trace of the target platform (in this case, windows) is present in the original query.

Finally, we can observe the matching behavior in case the queried software version is more recent than the one tracked in the database. This is actually a very frequent condition, as the CPE database is often incapable of keeping track with every new software release in a timely manner and instead is often trailing behind a bit. It is not uncommon for the CPE database maintainer to delegate the appropriate version update for when a corresponding CVE gets released. Of course, security analysts should instead try to stay in front of the news as much as possible.

As shown in Figure 3.3, the user receives no results if the specified version is not yet present in the database. Again, it could be argued that a retrieval system should output no results if it considers that a sample is indeed missing, but for the specific use case in which such a system would be utilized, this could result in a serious vulnerability problem.

3.3 CPE guesser

https://github.com/cve-search/cpe-guesser

The CPE Guesser can be executed completely locally but does not take software versions into account and it too implements an AND-based matching policy. As a result, more specific queries yield fewer results, as all specified components must be present in a match. This can limit recall, making it difficult to retrieve relevant CPE entries when dealing with incomplete or noisy input data.

As we can see in this example:

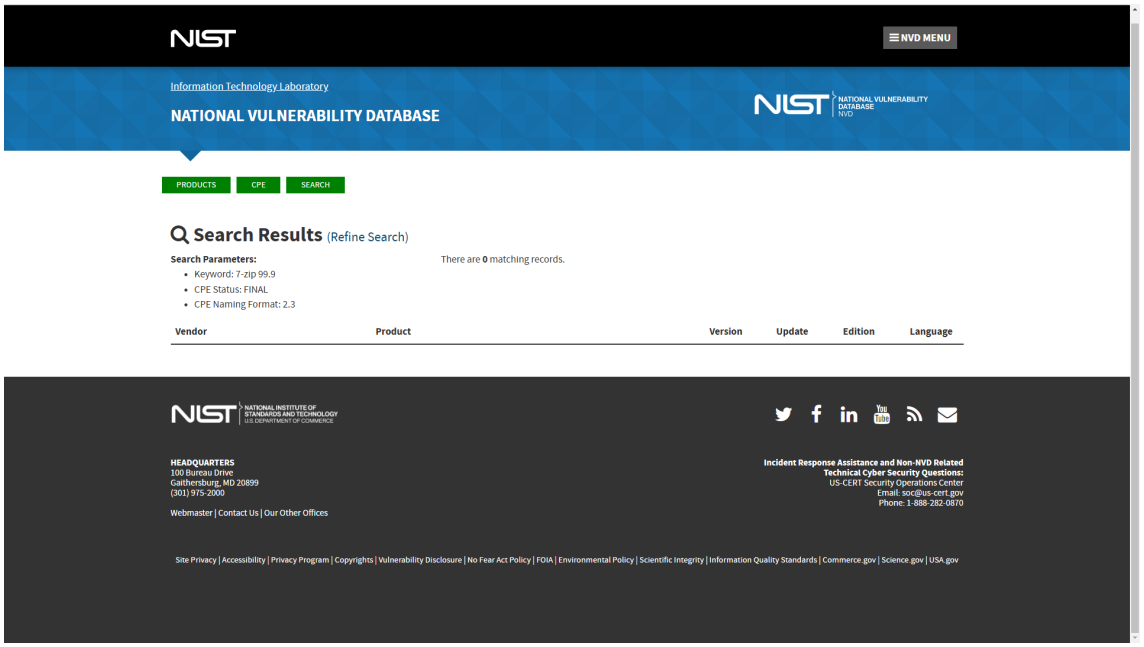


Figure 3.3: No results get returned for a yet-to-be released version of an existing product ('7-zip 9.99')

```
curl -s -X POST https://cpe-guesser.cve-search.org/search -d "{\"query\": [\"7-zip\"]}" | jq .
[
  [
    120218,
    "cpe:2.3:a:7-zip:7-zip"
  ],
  [
    115272,
    "cpe:2.3:a:7-zip:p7zip"
  ],
  [
    66076,
    "cpe:2.3:a:7-zip:7zip"
  ]
]
```

A correct set of results get returned, but by specifying the version too:

```
curl -s -X POST https://cpe-guesser.cve-search.org/search -d "{\"query\": [\"7-zip\", \"9.20\"]}" | jq .
[]
```

The number of returned CPEs drops to zero. This is undesirable, as more informa-

tion in the query, particularly if relevant, should not impact retrieving performances.

3.4 CPE search

https://github.com/ra1nb0rn/cpe_search

The CPE Search represents a significant improvement by introducing a term-frequency (TF)-based approach followed by a cosine-similarity evaluation, allowing for more flexible matching. A major advantage is its ability to search across different software versions, enhancing recall and relevance.

However, this method struggles with very simple queries. In such cases, cosine similarity can work against the user, leading to suboptimal results. This limitation is demonstrated in the following example:

```
./cpe_search.py -v -q "7-zip"
[]
```

actually brings no results, but internally we can examine the partial result:

```
[('cpe:2.3:a:7-zip:7zip:9.20:*:*:*:*:*:*:*',
   0.4765806309916866)]
```

which doesn't get output because the cosine similarity isn't above the one half threshold. If we extend the query by including the version as well, we obtain

```
./cpe_search.py -v -q "7-zip 9.20"
cpe:2.3:a:7-zip:7-zip:9.20:*:*:*:*:*:*:*
[('cpe:2.3:a:7-zip:7-zip:9.20:*:*:*:*:*:*:*',
   0.9940527304721751),
 ('cpe:2.3:a:7-zip:7-zip:9.20:*:*:*:*:windows:*:*',
   0.9334447481561471),
 ('cpe:2.3:a:7-zip:7-zip:-:*:*:*:*:*:*:*', 0.8988776375836615)
   ]
```

which correctly retrieves the right CPEs. This isn't ideal, as the most important feature of the CPE (i.e. the software name) gets matched but less important ones, like the version, are required to complete its retrieval. We would much prefer for the matching algorithm to somehow give more relevance to the more important parts of the query.

3.5 Summary

Each of the existing CPE matching approaches presents distinct advantages and limitations. The CPE Name search provides precise matches but struggles with

overly specific or generic queries, leading to either no results or an excessive number of candidates. The CPE Guesser simplifies the process by disregarding versions but enforces strict AND-based matching, which can overly narrow the results. The CPE Search improves flexibility with a TF-based approach and version-aware searching, yet it remains vulnerable to failure in simple queries due to the unintended effects of cosine similarity.

In conclusion, we can assess that the quality of all proposed solutions is directly correlated with the cleanness and precision of the provided data. The unrealistic expectations to meet this requirement in face of an unknown software stack format, means that a human operator is often required to form an ad-hoc query for each sample or parse the large amount of results, which greatly degrades the wanted level of automation. These challenges highlight the need for a more robust and adaptive solution capable of balancing precision, recall, and computational efficiency while facing very noisy inputs.

Chapter 4

An Information Retrieval approach

Information retrieval (IR) is the process of obtaining relevant information from a large dataset based on a user query[20]. Traditional IR systems, such as search engines, rely on indexing and ranking techniques to efficiently retrieve documents that match user intent. One of the core challenges in IR includes handling ambiguous queries, managing vast and evolving datasets, and optimizing search algorithms to balance both precision and recall, the two main relevant metrics, which are graphically represented in Figure 4.1.

In the context of CPE matching, well known IR techniques can be used as an automated retrieval method to help streamline the process of retrieving the most relevant CPE record associated to a software name.. However, several challenges arise due to variations in software naming conventions, versioning schemes, and vendor rebranding.

One of the fundamental issues in information retrieval stems from the gap between the retrieval need (what the user wants to retrieve) and the actual query (what the user actually asks). This has a corresponding problem in CPE matching. Ideally, a query should accurately reflect the intended software name, but in practice, it often contains other information coming from the asset inventory tool, which may contain noise such as extra metadata, file paths, or system-specific identifiers that degrade matching performance. As a result, traditional search methods may fail to retrieve relevant results, necessitating ad-hoc IR strategies that can handle noisy inputs and ambiguous queries while maintaining accuracy.

4.1 Terminology

We will employ some terminology taken directly from Information Retrieval Theory which is briefly introduced here and accompanied by its interpretation in the context of CPE matching.

The **information retrieval process**, in this context, is the act of retrieving the

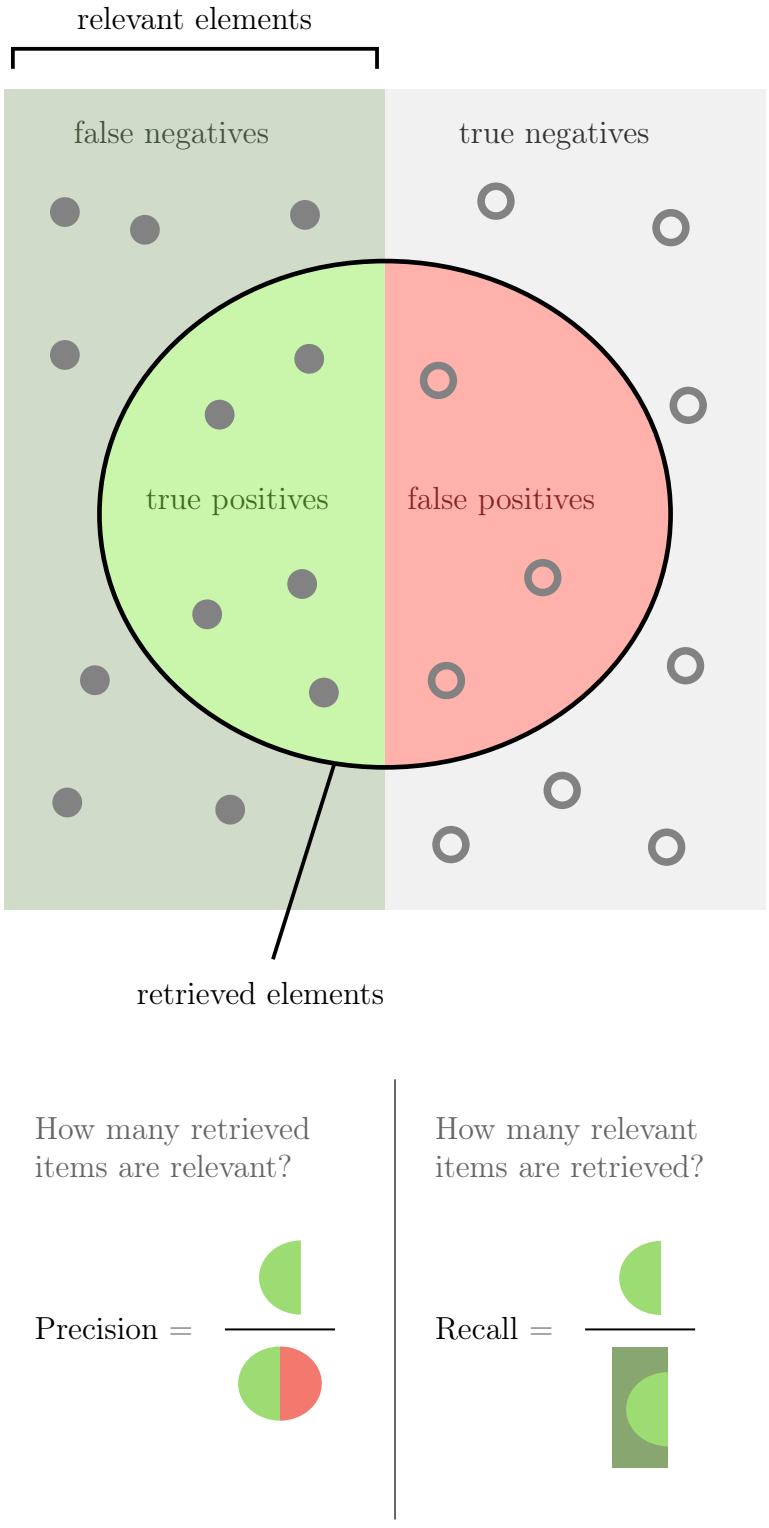


Figure 4.1: Precision and Recall By Walber - Own work, CC BY-SA 4.0

most relevant CPEs, given a query. A **query** is the string representation of an installed software which is present in an asset inventory. This may contain additional information like the vendor, the version, the target platform, and sometimes even reference links to the project website or repository, which contribute to the overall noise and must be specifically handled. The target retrieval information, a CPE in this case, is historically also called a **document**, and the set of all known documents is called a **collection** or a **corpus**. Although there exist multiple collections of CPEs, we will consider only the one officially maintained by NIST. Queries and documents are composed of **terms**: given that both are expected to contain only software names, we are not dealing with a specific language grammar, which removes the need of any natural-language stemming process. However, the concept of **stemming** can be defined in this context to represent a substring of a CPE object, which can be interpreted as a subset of the corresponding WFN containing only certain attributes. This can be useful, for example, to evaluate the matching performance on just the most relevant features. For example, a valid stem of the cpe

```
cpe:2.3:a:7-zip:7-zip:23.01:*:*:*:*:*:*:*
```

can be the substring containing only the vendor and product components:

```
7-zip:7-zip
```

4.2 Naive solution: fuzzy finder

An initial attempt at CPE matching involved using a fuzzy search within the string representations of all known CPEs. This method leverages string similarity metrics, like the Levenshtein distance [1] or the Jaro–Winkler distance [2], to identify approximate matches, allowing for minor variations in input queries.

However, this approach lacks sufficient control over what constitutes a valid match. String similarity alone does not necessarily align with retrieval needs, as it treats all character variations equally, regardless of their semantic importance. Additionally, increasing query specificity paradoxically reduces the likelihood of finding a match, as the algorithm rigidly enforces similarity thresholds. These limitations highlight the need for a more structured approach that considers both the hierarchical nature of CPEs and the inherent noise present in software inventory data.

4.3 TF-IDF based systems

The CPE guesser solution presented in section 3.4 shows promising results, but limits itself to only considering term frequency. A more advanced approach is to use the TF-IDF weighting scheme, which is a well-known technique in information retrieval to estimate the relevance of a document to a query.

[1] https://mi.mathnet.ru/dan31411
[2] https://eric.ed.gov/?id=ED325505

The TF-IDF weighting scheme is based on the idea that the importance of a term in a document is proportional to the number of times it appears in the document (term frequency) and inversely proportional to the number of documents containing the term (inverse document frequency). This approach helps identify terms that are both frequent in a document and rare across the entire collection, making them more likely to be relevant to a query. This is partially helped by the fact that, for marketing reasons, software names are generally chosen to be as distinctive as possible, which means that the most relevant terms are generally the ones that are the most rare.

TF-IDF methods make use of the binary relevance assumption, which states that relevance is a binary property of the document, given an information need only. Because an information retrieval system cannot know the values of the relevance property of each document, the information available to the system is at best probabilistic. This leads to the Probability Ranking Principle which asserts that if retrieved documents are ordered by decreasing probability of relevance on the data available, then the system's effectiveness is the best that can be obtained for the data[25].

4.4 Okapi BM25

The BM25 weighting scheme, often called Okapi weighting, after the system in which it was first implemented, was developed as a way of building a probabilistic model sensitive to term frequency and document length while not introducing too many additional parameters into the model [27]

Okapi BM25 is a ranking function widely used in information retrieval to estimate the relevance of a document to a given query by following a probabilistic relevance model. BM25 refines traditional term-weighting schemes by considering both term frequency and inverse document frequency while normalizing for document length. The core idea is that a term appearing frequently in a document increases its relevance, but with diminishing returns to prevent excessively long documents from being unfairly favored. At the same time, rare terms contribute more to relevance than common ones, ensuring that frequently occurring words do not dominate the ranking. By striking a balance between these factors, BM25 provides an effective and adaptable method for ranking search results, making it a strong baseline for many modern retrieval systems.

In the context of CPE matching, a CPE document $\bar{d}$ is said to belong to a CPE collection C. We may regard this as a vector $\bar{d} = (d_1, \cdots, d_V)$, where d_j denotes the *term frequency* of the jth term in $\bar{d}$ and V is the total number of terms in the vocabulary. In order to score such a document against a query, a term weighting function $w_j(\bar{d}, C)$ is defined which in the case of BM25 is

$$w_j(\bar{d}, C) := \frac{(k_1 + 1)d_j}{k_1((1 - b) + b\frac{dl}{avdl}) + d_j} \cdot \log \frac{N}{df_j}$$

where N is the number of documents in the corpus, df_j is the document frequency of term j, dl is the document length, $avdl$ is the average document length across the collection, and k_1 and b are free parameters. A k_1 value of 0 corresponds to a binary model (no term frequency), and a large value corresponds to using raw term frequency. b is another tuning parameter ($0 \leq b \leq 1$) which determines the scaling by document length: $b = 1$ corresponds to fully scaling the term weight by the document length, while $b = 0$ corresponds to no length normalization. Experiments have shown reasonable values are to set k_1 to a value between 1.2 and 2 and $b = 0.75$[20].

The document score is then obtained by adding the document term weights of terms matching the software name query q:

$$W(\bar{d}, q, C) = \sum_j w_j(\bar{d}, C) \cdot q_j$$

The BM25 term weighting formula has been used quite widely and quite successfully across a range of collections and search tasks. Especially in the TREC evaluations, it performed well and has been widely adopted by many groups.

4.4.1 BM25F

While BM25 operates on unstructured text, BM25F extends this approach by incorporating structured fields, allowing different components of a document to contribute differently to the final ranking score. This is particularly useful in the context of CPE matching, where software entries follow a predefined structure consisting of distinct fields such as vendor, product, and version. Rather than treating a CPE as a single block of text, BM25F enables a more refined approach by assigning different weights to these components.

By leveraging this structured representation, BM25F improves retrieval performance by reducing the impact of noisy queries. Software names extracted from asset inventories frequently contain additional metadata, such as platform information or distribution details, which do not always appear in the CPE representation. A structured ranking approach mitigates this issue by ensuring that the most relevant fields drive the retrieval process. This ultimately enhances ranking precision, leading to more accurate matches and fewer irrelevant results compared to simpler text-based methods like fuzzy searching or standard BM25.

In CPE matching, certain fields carry more significance than others. The vendor and product fields are the most critical, as they directly identify the software, whereas fields such as language or software edition are less essential but can still provide useful distinctions. Rather than completely excluding these less relevant fields, we assign them lower weights, ensuring that they contribute to the ranking only when necessary for disambiguation. This approach helps refine matches without allowing minor variations, such as language or edition, to dominate the retrieval process.

To compute the total relevance score, we follow the methodology outlined in [26], which highlights the limitations of naive linear combination of scores. Simply aver-

aging or linearly combining scores from different fields results in a potential violation of the nonlinear properties of term frequency weighting, as it fails to capture the varying importance of each component. Instead, a more refined aggregation formula is employed, preserving the individual contributions of each field while ensuring that the most relevant matches are ranked appropriately. This structured scoring method enhances retrieval effectiveness, improving both precision and recall in CPE matching.

Field Score Combination for BM25F

Specifically, we now consider a collection with a set of field types $T = \{1, \cdots, f, \cdots, K\}$. For example $f = 1$ may denote the Vendor, $f = 2$ Product etc. A structured document $\mathbf{d}$ can be written as a vector of K text-fields:

$$\mathbf{d} = (\bar{d}[1], \bar{d}[2], \ldots, \bar{d}[f], \ldots \bar{d}[K])$$

For example, $\bar{d}[1]$ would represent the vendor of the CPE d, $\bar{d}[2]$ the product, etc.

Each field $\bar{d}[f]$ may be seen as a vector of term frequencies $(d[f]_j)_{j=1..V}$, where V is the size of the Vocabulary containing all known terms. $\mathbf{d}$ is thus a matrix (a vector of vectors). The collection of structured CPE documents is then referred to as $\mathbf{C}$. Finally, in order to weight fields differently, the field weight are defined as a vector $\mathbf{v} \in \mathcal{R}^K$. The problem is therefore transformed into extending the standard ranking function $W(\bar{d}, q, C)$ into a new function $W(\mathbf{d}, q, \mathbf{C}, \mathbf{v})$

By merging all document fields into a non-structured form, by mapping documents as follows:

$$d := \bar{d}[1] + \cdots + \bar{d}[K]$$

we would not achieve the aim of exploiting structure.

Instead, each field type can be treated as a separate collection of (unstructured) documents and the standard ranking function can be applied to each collection separately

$$W(\bar{d}[f], q, C) = \sum_j w_j(\bar{d}[f], C) \cdot q_j$$

to obtain a linear combination of these scores using field weights:

$$W_1(\mathbf{d}, q, \mathbf{C}, \mathbf{v}) := \sum_{f=1}^{K} v_f \cdot W(\bar{d}[f], q, C)$$

While this approach, referred to as **(linear) combination of scores** appears simple, it presents some problems, namely the loss of the desirable nonlinear behavior over term frequencies.

Other variants such as [18] (called BM25FIC, field information content) exist, but have the key difference that the field weights are applied to each document separately rather than to the entire field, as normally done by BM25F where the

field weights are constant across documents. Although showing promising results, those alternatives make tuning parameters harder to understand and control.

The main contribution of [26] lies in a new proposed method, called **linear combination of TFS**, which satisfies the following requirements, that are absent in the linear case:

- **preserve term frequency non-linearity** which has been repeatedly shown to improve retrieval performance.
- **give a simple interpretation** to collection statistics and to document length incorporating field weights. Specifically, a field weight that is twice as large results in that field being twice as important.
- **revert to the unstructured case** when field weights are all set to 1.

With this approach, term frequencies of the different fields are combined by forming a linear combination weighted by the corresponding field weights:

$$\mathbf{d}' := \sum_{f=1}^{K} v_f \cdot \bar{d}[f]$$

and $\mathbf{C}'$ is the new collection of documents. The document scoring is then computed using the resulting term frequencies:

$$W_2(\mathbf{d}, q, \mathbf{C}, v) := W(\mathbf{d}', q, \mathbf{C}')$$

This method is equivalent to mapping the structured collection into a new non-structured collection with modified term frequencies which combine the original term frequencies in the different fields, weighted.

k_1 and tf

The k_1 parameter controls the non-linear tf function of BM25, but because the new method for BM25F substantially changes tfs, we may expect to change the optimal value of k_1. We can observe that by using the linear combination of frequencies with all field weights the same but not equal to 1 ($v_f = v \neq 1$, equivalent to multiplying all tfs by v), then we would obtain exactly the same results as the unweighted case by also multiplying k_1 by v. The optimal k_1 is therefore v times optimal k_1 for the unweighted case. Optimal b would be unchanged.

The average tf can therefore be use as a guide on how to change k_1. By considering the optimal values k_1^* and b^* for the unweighted case, we can derive

$$b = b^* \quad \text{and} \quad k_1 = k_1^* \frac{atf_{weighted}}{atf_{unweighted}}$$

where atf is the average term frequency.

This prevents the need to re-optimize the tuning parameters for different field weights.

4.4.2 The lack of a standard dataset

A significant challenge in evaluating matching performance arises from the inherent characteristics of software identification. The dynamic evolution of the software ecosystem, coupled with the proliferation of diverse implementations and versions for nominally identical software products, precludes the existence of a universally accepted, standardized dataset suitable for benchmarking name matching algorithms in this domain. Consequently, a custom-built evaluation dataset becomes a necessary recourse for performance assessment.

For this purpose, a web interface has been developed to make use of a standard implementation of BM25 in an interactive way to collect a dataset from real-world asset inventories. This dataset was then used to evaluate the performance of the BM25F implementation. The dataset was collected by manually selecting a set of software names from the asset inventory and querying the CPE collection to identify the corresponding CPE records. This process was repeated for a diverse range of software names, including both common and less prevalent entries, to ensure a comprehensive evaluation of the matching algorithm. It should be noted, however, that even with those considerations, this kind of evaluation is very prone to overfitting to the specific instance of asset inventory that has been used. A more extensive collection of humanly evaluated matches should be developed to account for a more diverse set of asset inventories and obtain a more generalizable result's description. The resulting dataset was then used to assess the accuracy and efficiency of the BM25F implementation, providing valuable insights into its retrieval capabilities and potential areas for improvement.

Chapter 5

Implementation

5.1 Workflow as a Continuous Process

In the ever-changing landscape of cybersecurity, the efficacy of any vulnerability management system depends on its ability to adapt and remain current. Static approaches to threat detection and mitigation are inherently vulnerable to obsolescence when facing rapidly evolving threats, daily discoveries of new vulnerabilities, and the constant flux of software deployments within organizational infrastructures. To deliver truly pertinent and timely CVE alerts, the developed system is developed around a continuous process. This ensures that the foundational CPE inventory, upon which the alerting system rests, remains constantly accurate and relevant. This, in turn, requires the system design to take a particular focus on the raw performance achievable, as this will be the main factor in determining the system's ability to keep up with the continuous flow of updates.

5.1.1 Two Streams of Continuous Updates: CPE Evolution and Inventory Dynamics

The persistent operation of the system is carefully orchestrated by two distinct categories of input events: **CPE updates** and **Inventory updates**.

A **CPE update** event signifies a transformation within the corpus of Common Platform Enumerations itself. This category encompasses two critical dimensions of CPE change: the creation of new CPEs and the modification or deprecation of existing ones.

CPE updates are sourced from the authoritative NIST's NVD CPE database, recognized as the standard reference for CPE information. Continuous ingestion of these updates ensures the system's foundational knowledge remains aligned with the established standards and best practices in CPE nomenclature.

In contrast to CPE updates, which address the evolution of the classification standard, **inventory update** events originate from client systems and represent changes within their specific technological environments. These events reflect mod-

ifications to the software and hardware assets actively deployed within a client's infrastructure. Inventory updates are triggered by an initial inventory submission or a subsequent inventory modification.

Client environments are dynamic, undergoing continuous changes through software deployments, upgrades, and decommissioning processes. Regular or event-triggered inventory scans from client systems generate inventory updates, reflecting these operational changes. These updates ensure the CPE inventory accurately represents the current software and hardware landscape within each client's domain, enabling the delivery of targeted and relevant CVE alerts.

The continuous processing of both CPE and inventory update events results in a dynamically maintained CPE inventory. By maintaining an accurate and up-to-date representation of both available CPEs and client-specific software deployments, the system facilitates the generation of vulnerability alerts characterized by:

- Timeliness: Alerts are generated based on the most recent vulnerability information and the latest CPE definitions, ensuring prompt notification of relevant threats.
- Relevance: Alerts are precisely mapped to the software deployed within each client's environment, minimizing extraneous alerts and focusing on actionable vulnerabilities.
- Actionability: By providing targeted and relevant alerts, the system enables users to effectively prioritize and address vulnerabilities impacting their specific technology stacks, thereby enhancing their overall security posture.

5.1.2 BPMN diagram

For a visual representation of this continuous workflow and the interaction of CPE and inventory update processes, a BPMN diagram was developed and is shown in Figure 5.1. This diagram provides a comprehensive overview of the system's operational flow and emphasizes its continuous and iterative nature and is here briefly discussed. The main objective of this diagram is to guide the subsequent implementation of such a process.

The main *CPE Matcher Framework* process is divided in 3 pool lanes:

- CPE Ingestion dedicated to keeping track of CPE Updates.
- CPE Matcher which performs any automatic or manually scheduled inventory match.
- Inventory Manager that keeps track of Inventory Updates

Two additional pool lanes are presented to expand upon the **Match Inventory** and **Optimizer** processes.

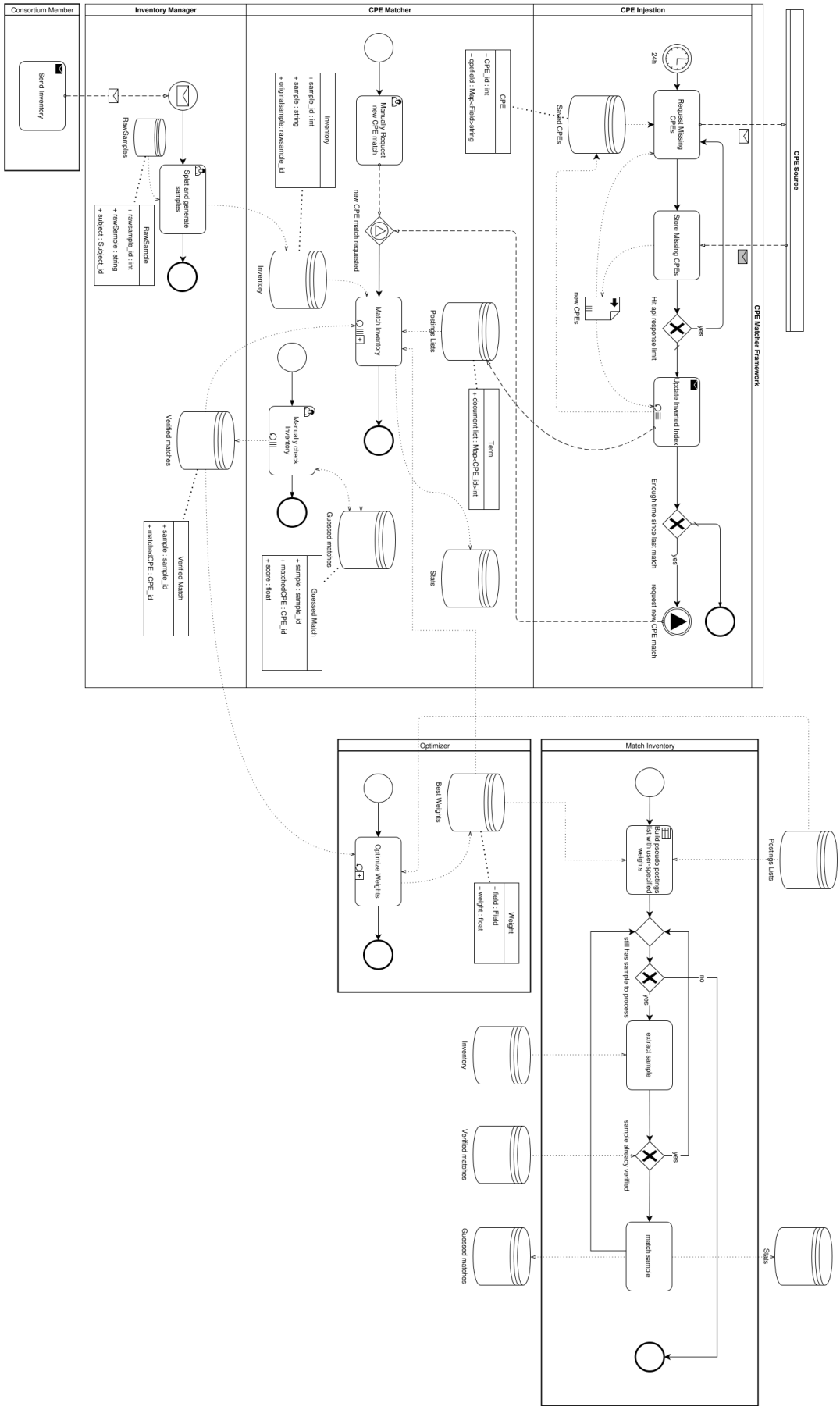


Figure 5.1: BPMN diagram of the CPE matcher process

CPE Ingestion

The retrieval of new and updated CPEs is generally scheduled to occur every 24 hours. The pagination behavior of the API (partially mitigated by a higher request rate if we dispose of an API KEY) requires performing multiple requests to the same endpoint until all updates have been downloaded. The same process can be performed for the first time to obtain an up-to-date version of the database from scratch. If the local inventory had an update, a new postings list is computed and a new request for a CPE match is sent to the matcher process. A posting list is a specific data structure used to represent an inverted index of the CPE database, that is, a map from any term to the list of all documents containing that term and its frequency.

CPE Matcher

A matching operation of all software inventories can be requested automatically after a local update of the CPE database, or manually if a user requires it. The Matching Inventory process makes use of the most recent posting list obtained by the local database and uses configuration-defined field weights to compute the pseudo-posting list containing updated term frequencies as per BM25F. All samples that have not been verified by a human have been checked against the CPE database to obtain the list of guessed CPEs. Those guesses can either be accepted to be used as the basis for an external alerting system or manually verified by a human.

Inventory Manager

Given the lack of assumptions that can be made about the asset inventory format used by different clients, each of these inventories must be handled specifically upon its reception. This is a process that should happen only once per inventory update or creation. A specific script is used to convert the input list of records to a list of samples with the particular requirement that each sample only contain one software name. Once those samples are created, they can be stored for later use by the matching process.

Match Inventory

The Match Inventory process highlights all the components necessary to perform a CPE match of an asset inventory. Specifically, the pseudo-posting list is computed each time a new matching process is requested, which allows users to tune field weights over different runs, if they so choose. The schema shows the main loop over the entire inventory, but of course the independence of each sample to the others allows the implementation to be easily parallelizable.

Optimizer

A final, optional, process considers the choice of field weights. Using BM25F, those become the main tuning parameters which affect matching performance and should therefore be chosen carefully. No specific optimization strategy is defined in this schema and is expected to be adapted to aimplementation's needs.

5.2 Python libraries

Okapi BM25 is a well known algorithm that has many implementations in different languages but not many allow for more advanced custom usages. We will present some existing alternatives for the python programming language.

The main implementation is a library called `rank-bm25` which only offers the following flavors of the algorithm:

- okapi bm25 The default algorithm
- bm25L This version addresses the problem observed by LV & Zhai that the document length normalization of BM25 (L_d/L_{avg}) unfairly prefers shorter documents to longer ones
- bm25+ A general solution proposed by LV & Zhai which lower-bounds the contribution of a single term occurrence: this solves the penalization of long documents which occurs not only in BM25 but other ranking functions too.

Another implementation comes from the scikit-learn library. Sklearn only implements `tfvectorize` which can be used as a starting point to implement any tf-based algorithm, but any more advanced matching algorithm is left to the user to be developed.

5.3 Custom implementation

The context in which the information retrieval algorithm is used is very specific and requires a custom implementation to fully exploit some characteristics of the data. For example, given the grouping expressions capabilities of CPEs, it would be beneficial if the ranking algorithm could perform part of those grouping operations to further refine the returned results. Wildcards, specifically, should be handled in a special way. Also, the required explainability for the final results imposes a more in-depth view of the algorithm's inner workings. As we have seen, some implementation of BM25 already exist online, but none that implement BM25F with the aforementioned requirements.

Regarding the performance objectives, we are in the specific case where a bunch of queries should be performed together in large batches, which leads us to prefer throughput over latency. We can easily observe that each query can be executed independently from each other, and so the system would greatly benefit from any

form of parallelism acceleration. Furthermore, the score for each term within a query can be computed out of order and independently of any other term before the final score aggregation. Therefore, it can be useful to reason about parallel execution units at the level of individual terms. For these reasons, Golang has been chosen for its relatively straightforward use of goroutines.

Goroutines are lightweight, user-space threads in the Go programming language, designed to enable efficient concurrency. Unlike traditional operating system threads, goroutines have minimal overhead, allowing the execution of millions of them within a single Go application. They are created using the `go` keyword, which initiates a function as a separate concurrent execution unit. The Go runtime manages their scheduling, dynamically multiplexing them onto available OS threads to optimize performance and resource utilization.

Goroutines facilitate parallelism by allowing multiple tasks to execute simultaneously, improving computational efficiency, particularly in multi-core architectures. They communicate via channels, which provide a safe and structured mechanism for inter-goroutine synchronization and data exchange, mitigating race conditions and deadlocks. Their lightweight nature and efficient scheduling make goroutines a powerful tool for writing scalable and parallel code with minimal complexity.

It can also be noted that NIST's CPE collection is large but not unreasonably huge. As of February 2025, the total number of non-deprecated CPEs is slightly under 1.3 million and the whole list of CPEs with their titles can fit in a 124MB plain-text file. Therefore, the whole retrieval process can be performed in memory, without an external db. This removes the need to implement a caching mechanism to obtain appreciable performances and generally simplifies the overall design.

5.4 Core Matcher Development

The first step in the development of the core matcher is the creation and maintenance of a local CPE database. This database is populated and updated based on data obtained from the National Institute of Standards and Technology (NIST) National Vulnerability Database (NVD) JSON API[1]. The data is retrieved following NIST's guidelines for CPE data management, ensuring consistency and accuracy in the local database. This approach aligns with established best practices for CPE data handling, facilitating efficient access and representation within the system[2]. CPEs are then filtered to remove deprecated entries, ensuring that only relevant and up-to-date CPEs are included in the local database. This filtering process is essential for maintaining the accuracy and relevance of the CPE inventory used for matching operations. The core matching component of the system is implemented using the BM25F algorithm, as described in subsection 4.4.1. In particular, we directly use the CPE attributes as document fields, with an additional field for the CPE title.

[1] https://services.nvd.nist.gov/rest/json/cpes/2.0
[2] https://nvd.nist.gov/developers/api-workflows

The title of a CPE is a common name for the software product and is always present inside NIST's database, sometimes even with its translation in multiple languages, if available. It has been verified that all known CPEs present in the database have an English title.

Field aggregation is performed based on the improved BM25F algorithm proposed by Robertson et al. [26], as described in subsection 4.4.1. This includes the k_1 and b compensation.

To manage concurrency efficiently and optimize resource utilization, goroutine pooling is implemented within the matching service. Goroutine pooling reduces the overhead associated with the frequent creation and destruction of goroutines, enhancing performance and scalability. The implementation comprises two distinct goroutine pools: one dedicated to processing incoming queries (referred to as `CPEWorkerPool`) and another for handling term processing tasks within each query (referred to as `TermWorkerPool`). This separation of pools allows for granular control over concurrency and resource allocation, ensuring optimal performance under varying workloads without saturating available resources. We should in fact be careful not to spawn too many threads, as the overhead due to their management may outweigh the benefits of parallelization.

By avoiding the need of an external database, we limit the latency due to IO operations and make the algorithm as cpu-bound as possible. For this reason, the best results are obtained by setting the total number of goroutines close to the number of available core processors.

Features

First of all, an input query awaits for an available goroutine from the `CPEWorkerPool` that will process it. The query is then stripped of any known uninformative symbols, like commas ',' and quotes '"'. Each character is converted to lowercase and a series of heuristics are applied to extract as much information as possible from the provided query. In particular, the system uses regular expressions to scan for a URL pattern and converts it into a list of terms: it removes the TLD (top level domain) and filters each domain, subdomain and path folder against a black list of uninformative terms (for example, 'www', 'wiki', 'github', 'sourceforge', 'project' etc.).

Then, versions are detected and parsed: in particular, semantic versions [3] and Debian's control files' versions [4] are identified using regular expressions. This removes unwanted data that usually relates to the package distributor, rather than the actual software. Versions are then expanded into all possible leading substrings.

For example, the string `2:21.1.4-2ubuntu1.7~22.04.5` gets recognized as

$$\underbrace{2}_{\text{epoch}} : \underbrace{21.1.4}_{\text{upstream version}} - \underbrace{2ubuntu1.7 \sim 22.04.5}_{\text{debian revision}}$$

[3] https://semver.org/

[4] https://www.debian.org/doc/debian-policy/ch-controlfields.html

and only the upstream version gets saved, which then gets expanded to the list of values `'21', '21.1', '21.1.4'`. In this way, a partial match only for the major part gets a lower score than matching all major, minor and revision parts.

Then another series of heuristics implemented via regular expressions are used to expand certain word patterns. For example, words containing a combination of letters and numbers get split by those groups. Also, special words like 'lib' or 'browser' are identified even when part of a longer sequence. This, for example, transforms `libxstream` into `xstream` and `torbrowser` into `tor`.

Optionally, an alias file can be passed as an argument to the matcher, which will act as a thesaurus of known synonyms. This allows users to define a set of words that will be expanded in this phase. For example, `'smb'` can be expanded into `'samba'`, `'kubuntu'` into `'kde ubuntu'` and also `'il tuo telefono'` can be translated into `'your phone'`. On one hand, this is a good mechanism to help the matcher system in case the original word is not present in the CPE dictionary, on the other hand, this is a tedious work to be done manually. Future developments could make use of more refined methods like language models to automatically obtain such a thesaurus.

Every term found inside a query is then dispatched to available goroutines in the `TermWorkerPool`, which will calculate the scoring function for that term and return a partial rank for every CPE in the dictionary.

Finally, all partial matching results are aggregated and post-processed to refine and organize the result set. Results having equal score are grouped by product and vendor, which prevents the same product having different version or edition to fill the entire ranking, and instead give space to a more diverse result set.

Within each product-vendor group, results are further sorted internally according to version information, prioritizing entries with more recent versions in descending order. This version-based sorting ensures that the system returns the most current and relevant CPE entry that is immediately before the one detected when multiple matches are found for the same product and vendor. A specific heuristic is incorporated to handle scenarios where version information is absent in the query. In such cases, CPE entries with a missing version field (represented by '-') are considered as a default match, provided no version-specific matches are identified. Alternatively, the CPE with a wildcard (represented by '*') in the version is used. This accommodates common user queries that may omit explicit version details while still returning potentially relevant CPE matches.

5.5 Web app for human disambiguation

The development of a web interface for human disambiguation and relevance feedback is a critical component of the system. This interface, shown in Figure 5.2, enables users to interact with the system, review matching results, and provide feedback to refine the matching process. The web app is designed to be minimal but intuitive and responsive, facilitating efficient interaction and enhancing user ex-

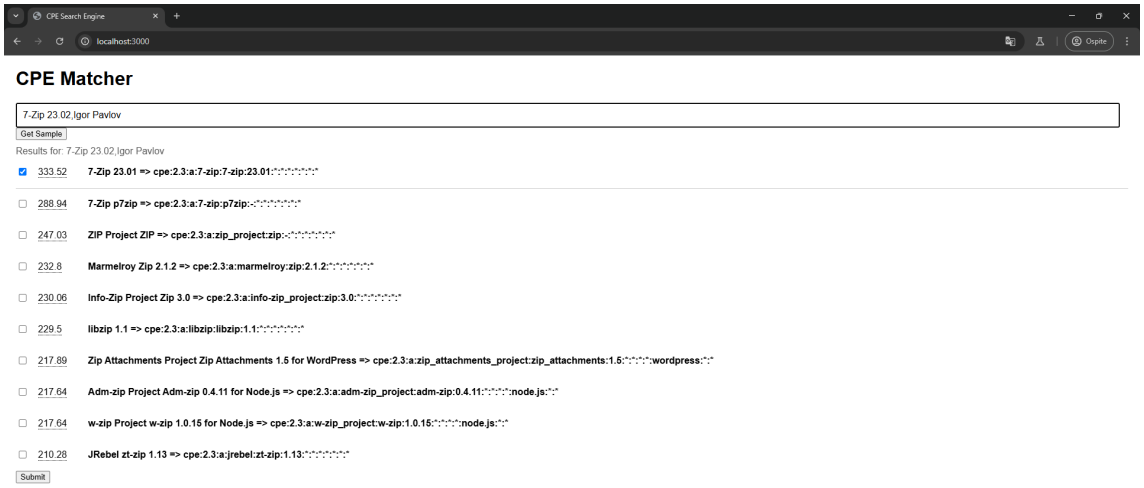


Figure 5.2: Web interface for the algorithm

perience. The same web application is used for both relevance feedback and manual matching of samples to CPEs, providing a unified interface for these distinct but related tasks.

The interface presents a search bar that can be used by the user to perform a CPE match. On each text modification, the result list gets updated. The `Get Sample` button just under it retrieves a random sample from the database of unchecked samples and replaces the current query with it. The list shows the top K results from the matching algorithm in descending order. Each result is composed of the achieved result with the returned CPE in the form of `Title => CPE Uri`. Results are also automatically pre-selected using the matcher strategy, this allows users to quickly see which results would have been returned as a match, but also which are the runner-ups. A vertical line is also inserted between the two results having the largest difference in scores. This 'largest gap' information is used for a more refined filtering strategy, as described in section 6.1.2

The web app is implemented using a thin fron-end layer implemented with Preact[5] (a minimal subset of the more popular React-js framework), and the backend is written in Go, which communicates with the core matcher executable to perform matching operations. The backend manages the interaction between the user interface and the matching service, handling requests, processing responses, and managing user feedback. The backend is designed to be lightweight, and saves the results that have been analyzed to a document collection handled by MongoDB[6], a NoSQL database that allows for efficient storage and retrieval of JSON-like documents.

[5]https://preactjs.com/
[6]https://www.mongodb.com/

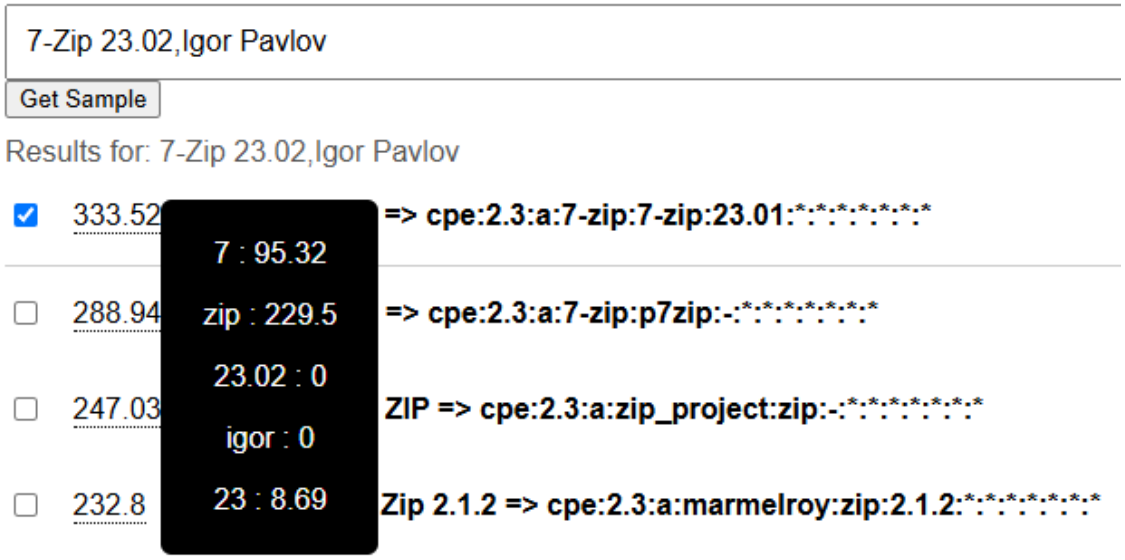


Figure 5.3: By hovering over a score, we can get each term's contribution to the final score

To aid the human evaluator in understanding why a certain match has occurred, an explanation of each term's contribution to the total score is saved. For example, in Figure 5.3 we can clearly see why the first result obtains such a score. Only terms present in the vocabulary are shown, as they are the only one that can contribute to the score. We can observe how a large part of the score is given by the term 'zip', which is clearly the most important part of the query. This example can also clearly demonstrate the advantage derived from using the field-extension version of BM25: the number '7' is in fact very commonly used inside the 'Version' field but because it appears in the Product and Vendor names of the software '7-zip' it achieves a much higher score with respect to 23, which, in this instance, is present only in the version. We can also note how the algorithm handles versions. By recognizing the common pattern of Semantic Versioning, each version string gets expanded into all possible substrings starting from the beginning and ending at a dot ('.').

5.6 A common format

The system is designed to handle input data heterogeneity by defining a minimum common format for input samples. This common format consists of pure strings, ensuring compatibility with a wide range of input sources and formats. To accommodate diverse input data types, specific translators are implemented to convert various file formats into the common string format. For example, translators are developed to process CSV files exported from different asset inventory systems, such as GLPI[7] and Wazuh[8], into the common string format. These translators extract

[7]https://glpi-project.org/
[8]https://wazuh.com/

relevant information from the input files and convert it into a standardized string representation, enabling seamless integration with the matching system.

This is the only non-automatic part of the system, as it requires a human to define the mapping between the input fields and the common format. Although it should be noted that such a procedure should be performed only once for each input source, and the system can then automatically process new input files using the defined mapping.

In case of a csv file, for example, it would be sufficient to concatenate the fields of interest with a separator, such as a comma, to obtain the common format.

The input rows are further processed using a splatter mechanism to generate multiple samples from a single record. It is in fact possible that a single record contains multiple software components, for example by having a common vendor and product name, but a list of installed versions. Each sample is designed to contain at most one CPE, although multiple CPEs may be included to increase the likelihood of matching the correct one. This approach enhances the system's flexibility and robustness, enabling it to handle complex input data structures and diverse software naming conventions effectively. By standardizing input data into a common format and implementing a splatter mechanism to generate multiple samples, the system can efficiently process a wide range of input sources and formats, enhancing its usability and adaptability.

Let's take a look at a real-world example of a CSV row, taken from GLPI that could be used as input for the system (the first header line of the csv is included for understandability):

```
1  Product,Entity,Vendor,Versions - Name,Versions - Operating
   Systems,Number of Installations
2  7zip,REDACTED,Ubuntu,23.01+dfsg-11<br>23.02+dfsg-11<br
   >23.01+dfsg-11,Ubuntu 24.04 LTS<br>Ubuntu 24.04.1 LTS,3
```

First, we would extract the relevant fields, in this case, the product, vendor, and versions. Then we would concatenate them to obtain the common format:

```
1  7zip Ubuntu 23.01+dfsg-11<br>23.02+dfsg-11<br>23.01+dfsg
   -11
```

Finally, we would splatter the record to obtain multiple samples, one for each version:

```
1  7zip Ubuntu 23.01+dfsg-11
2  7zip Ubuntu 23.02+dfsg-11
3  7zip Ubuntu 23.01+dfsg-11
```

In this case, we would benefit from skipping redundant work, therefore a final

filter for duplicates is applied to the samples, and we obtain the final list of samples to be analyzed.

```
    7zip Ubuntu 23.01+dfsg-11
    7zip Ubuntu 23.02+dfsg-11
```

This can be resumed in the following bash script:

```
    awk 'NR>1 {print $$1, $$3, $$4}' FS=',' OFS=' ' glpi.csv \
    | python3 splatter.py --fs=',' --rs='<br>'\
    | sort -u > input.txt
```

Where the splatter.py script is a simple python script that takes care of the splatter operation.

```
from itertools import product
import argparse

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Splatter')
    parser.add_argument('--fs', type=str, help='Field
    Separator', default=',')
    parser.add_argument('--rs', type=str, help='Record
    Separator', default='<br>')
    args = parser.parse_args()
    while True:
        try:
            in_str = input()
        except EOFError:
            break
        if args.rs not in in_str:
            print(in_str)
        else:
            versions_per_field = [field.split(args.rs) for
    field in in_str.split(args.fs)]
            arr = [args.fs.join(x) for x in product(*
    versions_per_field)]
            for res in arr:
                print(res)
```

5.7 Field Weights optimization

Individual field weights are the main parameters governing the system's capability to retrieve the correct CPE. It is therefore necessary to find suitable values specifically optimized for CPE matching.

The final program allows users to pass as command line arguments the exact field weights to be used for a specific run. Specifically, these are the available fields that can be tuned (obtained through the `-h` flag):

```
-edition float
      Weight for field edition (default 1)
-language float
      Weight for field language (default 1)
-other float
      Weight for field other (default 1)
-part float
      Weight for field part (default 1)
-product float
      Weight for field product (default 1)
-sw_edition float
      Weight for field sw_edition (default 1)
-target_hw float
      Weight for field target_hw (default 1)
-target_sw float
      Weight for field target_sw (default 1)
-title float
      Weight for field title (default 1)
-update float
      Weight for field update (default 1)
-vendor float
      Weight for field vendor (default 1)
-version float
      Weight for field version (default 1)
```

These form a 12 dimensional search space which quickly becomes unmanageable for an exhaustive search.

In general, optimizing standard IR measures, however, is not easy: they are very expensive to evaluate, they have local maxima and plateaus, they are not smooth and they don't have gradients[24].

Various optimization techniques have been used in literature. For example, [25] proposed a method called **Robust Line Search** which is a Greedy algorithm with promising results, but its recursive nature makes it not practicable, as noted by the authors:

> [...] this approach remains exponentially expensive with respect to [the number of dimensions] n because of its recursive nature, and therefore it is not practicable for large n (e.g., n $>$ 3)

Another approach proposed by [25] consists in the approximation of rank-dependant relevance functions such as NDCG by a function with known gradients, which can then be used to perform known machine learning techniques.

Field	Value
title	17.0
vendor	48.5
product	68.5
version	1.5

Table 5.1: Final field weights optimized via genetic algorithms

This study is primarily focused on the applicability of the BM25F approach on the problem of CPE matching and therefore a more general approach, known to work over a vast range of problems, has been used. Field weights have been optimized using a genetic algorithm.

Genetic Algorithms (GAs) are a type of optimization technique inspired by the principles of natural selection and evolution. They are used to solve complex problems by mimicking the process of natural evolution, applying concepts such as mutation, crossover (recombination), and selection.

GAs operate on a population of potential solutions, evolving them over generations to find an optimal or near-optimal solution.

In this specific instance, a population of 10 individuals is evolved through 100 generations with a mutation probability of 10%, a crossover probability of 50% and with each individual's genome composed of only 4 characteristics: the field weights for the title, vendor, product and version. All other field weights have been left to 1, as we're not interested in the absolute value obtained by each weight, but instead to the ratio between each pair. Also, these 4 fields are considered to be the most important in the identification of a CPE, as the majority of other fields is often empty or composed of an uninformative '*' wildcard.

The training fitness function is reported in Figure 5.4. Training has been stopped after 100 generations. No particular sign of plateauing has been shown, which suggests the possibility that a more extensive optimization procedure may result in better results.

The final result converged to the field weights presented in Table 5.1 (all other fields are considered to be equal to 1)

We can observe that the product field has the highest impact in performance as it is clearly the one that drives the retrieval system for the correct CPE. Secondarily, the vendor field is still relatively important to distinguish between similar products and finally the title field has a lesser impact, as it often contains all the previous information and therefore has a higher risk of degrading performance if it is set too high. We can also note that the version field reached a very low value. Intuitively this makes sense, as it is the one that has the least probability of identifying a CPE because many software may share the same version across different vendors. It becomes useful only as a disambiguator, once a set of plausible CPEs has been selected by the other fields.

Due to the long times required to process each evaluation, some considerations

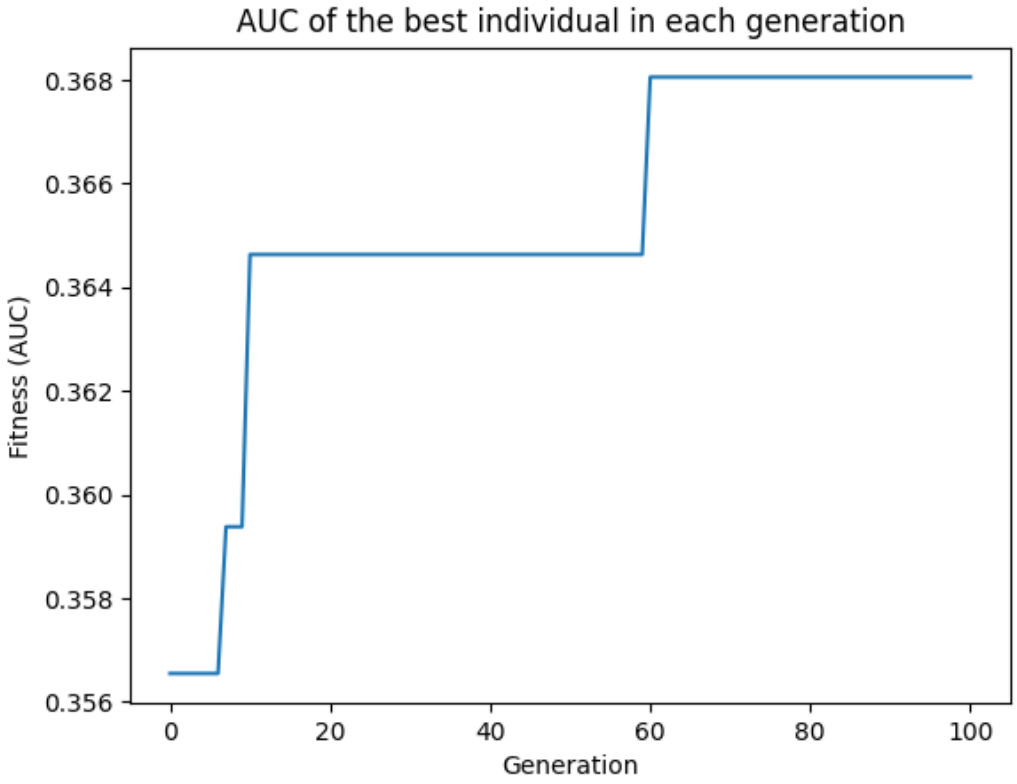


Figure 5.4: Fitness Evaluation during Genetic Algorithm optimization

have been made. First of all, results for a given genome (which is a combination of field weights) are cached to prevent having to recompute a previous result. This is of course permitted by the deterministic nature of the algorithm, which will always return the same metric for a given combination of field weights.

Secondarily, testing is performed at the end of training, rather than during training. This makes it harder to identify the presence of overfitting, but it is a compromise chosen in favor of a faster training, which, in turn, can be made more in depth.

Testing is performed on a separate fraction of the dataset that has been kept off during training and is now used to compare the BM25F algorithm with optimized field weights with a default BM25 one, which is achieved with the same program by setting all fields equal to 1.

In Table 5.7 we can see the achieved precision and recall achieved by both the default unweighted case and the BM25F case with optimized weights. These metrics are reported over a list of 5 different K levels, which represent the precision and recall achieved by considering only the top K matches. We can see that the optimization of field weights results in a total improvement across all K levels. Finally at the bottom are reported the area under the curve (AUC) too of the corresponding precision-recall curves. Again, we can see an improved obtained in the optimized case, making BM25F a sensible choice for the CPE matching task.

Table 5.2: Comarison between the default BM25 and BM25F with field weights optimized via GA.

K	Unweighted (BM25)		Optimized (BM25F)	
	Precision	Recall	Precision	Recall
1	0.771929825	0.723684211	0.815789474	0.76754386
2	0.486842105	0.881578947	0.495614035	0.894736842
3	0.352339181	0.928362573	0.361111111	0.947368421
4	0.277046784	0.937134503	0.283625731	0.951754386
5	0.239766082	0.960526316	0.243274854	0.973684211
AUC	0.4561		**0.5024**	

Chapter 6

Results

6.1 Matching performances

The evaluation of the CPE matching system is crucial to assess its effectiveness in generating accurate and timely CVE alerts. The primary metrics for evaluating the system's performance are precision and recall. While both are essential, recall is of paramount importance in the security context. False negatives, where the system fails to match a software name with its correct CPE, can lead to missed vulnerability alerts, potentially exposing systems to exploitation. In contrast, false positives, where the system incorrectly matches a software name with a CPE, are less critical.

The goal is to maximize recall while maintaining an acceptable level of precision. By focusing on maximum recall and analyzing the corresponding precision, we can effectively assess the system's suitability for delivering timely and relevant CVE alerts. We will explore different configurations and algorithms to identify the point at which recall is maximized. At this point, we will analyze the corresponding precision to understand the trade-off between comprehensive coverage and alert relevance.

In the context of CPE matching, we define precision and recall as follows:

- Precision: Out of all the CPEs the system matched to a given software name, what proportion were actually correct. High precision means fewer incorrect CPE matches are suggested.
- Recall: Out of all the correct CPEs that should have been matched to a given software name, what proportion did the system actually match. High recall means the system successfully identifies most of the relevant CPEs.

A false negative in our system occurs when the system fails to match a software name with its correct CPE, leading to a missed CVE alert. This is a critical failure because it leaves the user unaware of potential vulnerabilities affecting their systems. In contrast, a false positive occurs when the system incorrectly matches a software name with a CPE. This leads to sending an alert that is ultimately not relevant.

While both types of errors are undesirable, the consequences of false negatives are far more severe. Missing a critical vulnerability alert can have significant security ramifications, potentially leading to exploitation and breaches. False positives, on the other hand, are less damaging. They might cause temporary inconvenience or alert fatigue, but they do not inherently create security vulnerabilities. In essence, it is better to lean on the side of caution and generate some false positives than to miss crucial vulnerabilities.

Therefore, while we prioritize recall, we also aim to achieve a reasonable level of precision to maintain user trust and the long-term usability of the alerting system. The goal is to strike a balance where we maximize vulnerability coverage (high recall) while minimizing unnecessary noise (acceptable precision).

We will now discuss the achieved performances of the developed algorithm in term of how well it can match a given software name. This task is particularly difficult for multiple reasons. The lack of a standardized dataset means that a limited number of samples are available to extract information. Furthermore, the breath of the underlying software name distribution is pretty limited, given that it comes from a small number of asset inventories. Another problem that makes it more difficult to compare this information retrieval system with those already present in literature arises from the very specific context in which it is applied. Common search engines often deal with a very large pool of candidate documents, often with a varying degree of relevance. By contrast, in CPE matching, one software name should ideally match only one CPE, and its relevancy is mostly a binary property. It is possible that a human evaluator defines more than one CPE as relevant, but those cases should be considered as rare and more of a byproduct of an under-specified query. Finally, information retrieval systems are often assumed to be used in front of a human user, which then selects retrieved results with higher probability if they are ranked higher in the final list. CPE matching, in the other hand, should be considered as much as an automatic process as possible. Therefore, once a certain criteria is chosen to select which CPEs should be returned, it is no longer important the original order in which they originally appeared. For this reason, we will first analyze the matching performance of the final algorithm using rank-based metrics, in order to more easily get a sense for the base ranker performance, and subsequently we will employ set-based metrics to assess the final performance obtained by the automatic matching process by considering various filtering strategies that have been employed to improve overall performances.

6.1.1 Ranking based evaluation

For the ranking-based evaluation we employ the precision-recall curve.

A precision-recall curve is a per query plot of the precision (y-axis) and recall (x-axis) for different k-levels. The area under the curve (AUC) is a measure of the overall performance of the system. A higher AUC indicates better performance, with a maximum value of 1.0 representing perfect precision and recall. By examining

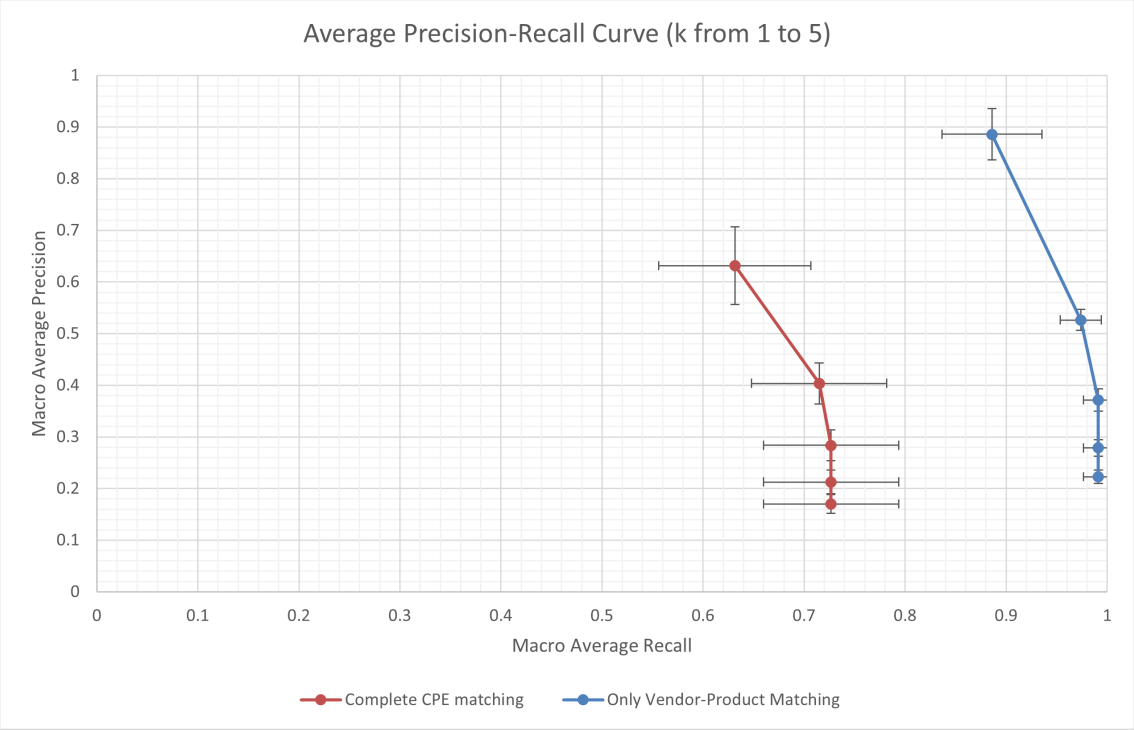


Figure 6.1: Precision-recall curve for the CPE matching system. Comparison between a full match. Comparison of the Precision-recall curve between a full match and

the curve, we can identify the optimal threshold that maximizes both precision and recall. This threshold represents the best balance between alert relevance and coverage, ensuring that the system generates timely and accurate alerts while minimizing false positives and negatives.

To evaluate the overall performance of the system across multiple queries, we calculate the macro-average precision and recall at different k-levels. This allows us to assess the system's performance in a more general context, providing insights into how well it performs across a range of queries and thresholds.

In particular, the two curves shown in Figure 6.1 present two different ways in which a CPE is considered to be a correct match. Exact matches require for the entire CPE to be matching, so all CPE fields like vendor, product, version, edition and so on should be exactly equal to those reported by the human evaluator. The second line only requires for the vendor and product fields to be equal. The reason for this distinction comes from the way in which the testing database has been constructed. Following the pseudo-relevancy strategy, the same retrieval algorithm has been used by a human evaluator, which given a certain query, would select which of the returned records are actually relevant. This implies that the testing dataset is dependent on the time in which it has been generated. Some of the software that were used as samples did not yet have an updated version in the official CPE database. The final retrieval system should ideally return the exact CPE or

alternatively its most recent version, therefore the human evaluators were instructed to still consider an out-to-date version as a match if both vendor and product were relevant. During subsequent performance evaluations, however, the CPE database used was newer than the one used during the testing dataset creation. This leads to the possibility that a newer CPE, which would be different than the one found by the human but a better match for a given query, would result incorrect if we were to consider a full CPE equivalence.

Precision recall points are considered for a varying number of results returned: considering only the top 1 result would be associated with k=1, top 2 results would be k=2 and so on. Precision and recall at each value of K are a property of each query. To obtain a general overview of the global retrieval performance across multiple queries, a macro average is performed for each k level. It is important to note that values are normalized to their recall level before being averaged. This means that in the rare case in which a query has a recall base of 3 (i.e. 3 CPEs are considered relevant) and, for example, we consider K=1, if that retrieved result is relevant we would consider recall= 1, instead of the more naive recall=1/3. The reason behind this normalization is to prevent statistics over lower values of K to be skewed towards worse results than they actually are. Indeed, matches with a high recall base shouldn't be penalized only because a lower value of K has been chosen.

We can clearly see the common pattern of a precision recall curve: with an increase in k we observe both an increase in the general recall and a decrease in the general precision. This is already a clear indicator that by considering more results we have a net effect of polluting the returned set of CPEs. If a good match is to be found, it generally happens in the first results, otherwise the majority of returned records is not relevant. By considering a larger number of records (corresponding to a higher K value) recall tends to be the same for the two approaches. Considering the exact match, we can make some considerations on the 90% confidence intervals that are shown on both axis. We can observe that the variation in precision shrinks with an increasing K. This is again explainable by the very small recall base of each query: by considering more and more results, the precision tends to be equal to 1/n. On the contrary, recall generally maintains the same dispersion. This is implied by the fact that the wanted CPEs are either immediately found or they are more likely never returned. Therefore queries are almost always completely matched or completely failed. Finally, we can assess that the reason the dispersion in recall shrinks in the case of partial CPE match is only due to the fact that results are being squashed to the optimal value of perfect recall, i.e. recall=1. By observing the overlapping of corresponding confidence intervals between the two curves, we cannot assess that any of the two curves has a better precision than the other, with a confidence of 90%. On the contrary, recall does not present any overlap up until $K = 5$, which indicates that by only considering a match over the product and vendor fields it is much more probable to retrieve an expected CPE, even if the number of wrongly returned CPEs remains almost the same. Again, this is expected, as we are dealing with queries having a very low recall base: if a match is

immediately found after the first result, any subsequent CPE will almost certainly be wrong.

6.1.2 Set based evaluation

The set-based evaluation is performed by considering the top K results for each query and calculating the precision and recall of the returned set. The precision is calculated as the number of correct CPEs in the returned set divided by the total number of returned CPEs. Recall is calculated as the number of correct CPEs in the returned set divided by the total number of correct CPEs in the dataset. The F1 score is the harmonic mean of precision and recall, providing a single metric to evaluate the system's performance. A higher F1 score indicates better performance, with a maximum value of 1.0 representing perfect precision and recall. The F1 score is a useful metric for evaluating the system's overall effectiveness, providing a balanced assessment of precision and recall. By analyzing the F1 score at different K levels, we can identify the optimal threshold that maximizes both precision and recall, ensuring that the system generates timely and accurate alerts while minimizing false positives and negatives.

Empirically guided improvements

In order to improve the quality of returned matches, some additional filtering techniques have been tested and are now discussed here.

After their discussion, a final comparison is made against the base case to evaluate their improvement.

From bag of word to term position importance

BM25 and all its derivations, included BM25F, are considered to follow a *bag-of-word* model. This means that documents and queries are interpreted as a set of terms with an associated frequency but the order of which is unknown inside the original string. Any permutation of terms is therefore equivalent to the algorithm and will always result in the same output.

In the context of CPE matching we can make use of the information gained during training (as discussed in section 5.7) to aid the retrieval system with its task. Specifically, we can observe that the product field is by far the most important feature of a CPE, as it is the one that more clearly identifies a software name. This is expected, as this is commonly used as the only information to identify a product.

We can also note that many asset inventory tools already have the information of what part of a software name corresponds to each field. GLPI, for example, can export its result as a csv containing columns like the ProductName, the ProductVendor, the ProductVersion and so on. If we assume that the user scripts used to convert all asset inventories into a list of plain text samples always have a 'Product

Name' feature and put it in front of the sample, we can make the first term of the query weight more in the retrieval process.

Specifically, now the system keeps the knowledge of the order in which each term of a query appears. If the first term, which ideally should represent the software name, doesn't appear in a specific CPE (i.e. its contribution to that document is 0), then the total score for that document gets penalized (for the experiments that were done, the final score was halved).

This should ideally remove any ambiguous matches which obtained a high score because of non interesting terms, meanwhile CPEs that contain the first term have their score slightly boosted, which in turn makes them more likely to appear in a higher position in the ranking.

From K levels to maximum separating gap

A good method to improve precision of the final result would be to exclude any result that we consider non relevant, even if they appear highly in the final ranking. Because software names are expected to match with a very small number of CPEs (most of the time, only one), any result after a correct match should not be considered.

We can exploit the final score of a given CPE as a relative evaluation of the relevance of that document with respect to the original sample provided.

By clustering the final ranking into two groups: CPEs having a high score and those having a lower one, we can try to consider results coming only from the former set.

In practice, this can be done pretty efficiently because results are already sorted. It is in fact only a matter of finding the largest difference in score between two consecutive results, which is also called the *largest gap*.

Instead of limiting the number of results to a fixed amount per query, we can return the top 10 results, find the largest separating gap and finally returned only CPEs coming from the group above the largest gap.

Experiments have also shown that most of the time, if a correct CPE is known to be present in the dataset, the group of CPEs above the largest separating gap has size of 1 and only contains the expected CPE.

For this reason, the web app described in section 5.5 shows the largest gap as an horizontal line dividing the ranking into two groups. This visually helps a human evaluator to better understand to which group each CPE of the ranking belongs to.

Limit n results in above gap group

We can further refine the largest separating gap approach by considering the fact that most software names are expected to match with only one CPE.

In those cases where a software name is ambiguous or not present in the database, the approach of maximum separating gap may yield too many false positives, which greatly pollute the quality of the returned CPEs. In the worst case scenario, a single

sample which is expected not to have any corresponding CPEs may return up to K unrelated CPEs.

As a precaution, it is then possible to limit the number of returned CPEs for a given query in case the group of results above the largest separating gap becomes too large. This number is defined as n.

6.1.3 Evaluation of all filtering strategies

Figure 6.2 compares the F1 score of all strategies at different values of k, from 1 to 5.

The F1 score is the harmonic mean between precision and recall, which allows to get a sense of both metrics with a single number and is often used in Information Retrieval to assess the goodness of a retrieval strategy.

In particular, the *default* strategy is the one consisting of only BM25F with optimized field weights as described in section 5.7, without any filtering strategy. We can clearly see that the F1 score quickly decreases with an increase in K. This is primarily due to a decrease in precision because if a correct match is found within the first result, recall immediately jumps to 1 and cannot possibly improve further. Meanwhile, by considering more and more results, the result set gets mainly polluted by unrelated CPEs, which worsen performances.

The *TI* strategy stands for Term Importance and is the one that penalizes results having the first term of the query not contributing to the overall score. The effect of the strategy appears almost as a constant benefit to the F1 score, with a vanishing return associated to an increase in K. This can be attributed to an increase in recall, as this strategy has a positive impact in quickly finding the relevant CPE and more frequently returning it as the first result. Afterwards, the same principle of polluting precision due to an increase in the result set's size applies, which makes the F1 score approach the same value as in the default case with a higher K value.

On the contrary, the *gap* strategy, which implements the maximum separating gap filtering approach, seems to have an opposite effect to the F1 score. At $K = 1$, we observe the same F1 score as in the default case, but instead of decreasing, it remains almost constant when K is increased. This is because this strategy works almost exclusively on the precision metric. This strategy works as a sort of hard limiter on the number of returned results. This means that if the first results obtain a much higher score with respect to the rest of the ranking, then we will almost certainly never return any other CPE, regardless of K. In a sense, if the retrieval system is sure about a result set, it will continue returning the same set of CPEs, even when asked to work with a higher value of K.

Finally, by observing the almost orthogonal behavior of those two strategies, we can evaluate the effect of having both of them applied. Indeed, we can observe that the *gap+TI* strategy, which implements both the maximum separating gap and term importance filtering, is capable of achieving the best of both worlds. At a low level of K (like $K = 1$), we obtain a boost in recall, given by the term importance, and at

the same time, with an increase in K, we maintain high precision, which translates to a very high F1 score across the entire range of K.

Figure 6.3 shows the final Precision and Recall achieved at different K levels by applying both strategies.

The final step would be to select an appropriate n value as a limit for the maximum separating gap strategy. For this aim, we can employ the generalized definition of an F score, also called F_β score. Particularly, F_β uses a positive real factor β, where β is chosen such that recall is considered β times as important as precision. The complete formula for computing the F_β score from precision and recall is

$$F_\beta = \frac{\beta^2 + 1}{(\beta^2 \cdot \text{recall}^{-1}) + \text{precision}^{-1}} = \frac{(1 + \beta^2) \cdot \text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

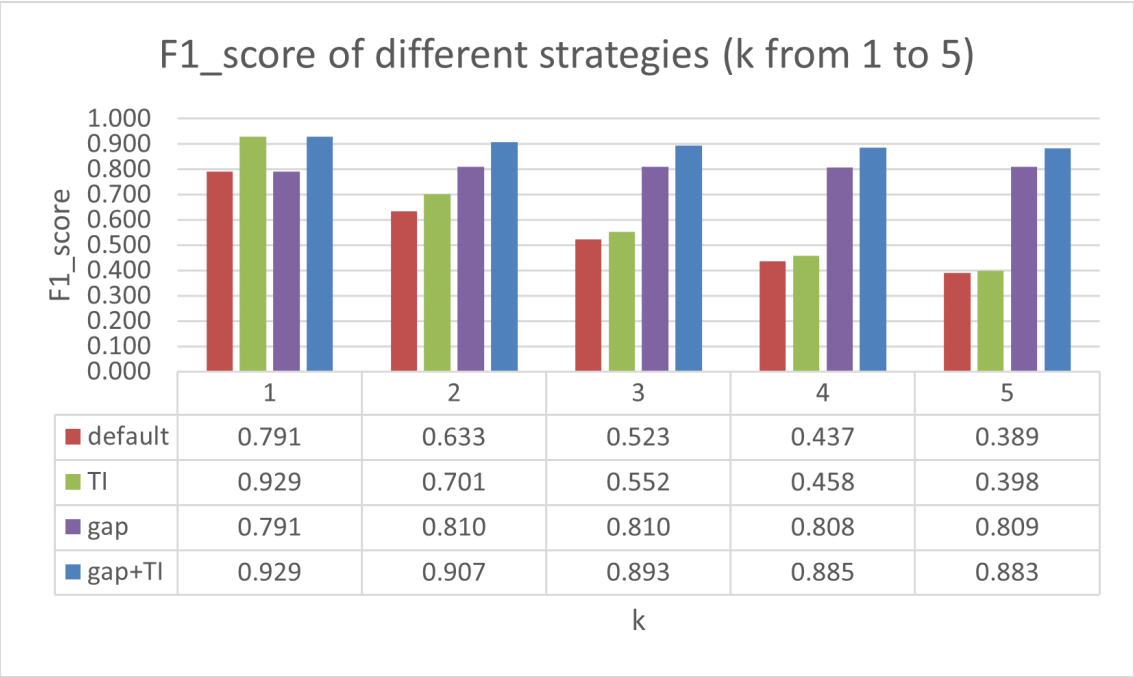
Figure 6.4 then shows a comparison for the final gap+TI strategy performance (as in Figure 6.3) but now with a varying β value. The case with $\beta = 1$ is exactly the one we obtained while computing the F_1 score. By considering a $\beta < 1$, we're implicitly analyzing the case in which we're more interested in precision rather than recall. Of course, then the highest F score is obtained with a very small value of K, which is expected due to the very small recall base of queries: any result other than the first one has a very high chance of being unrelated. In the context of software identification, and more specifically when the objective is to perform filtering of incoming CVEs, the risk of returning a false positive is outweighted by the cost of missing an alert, which is a false negative. Recall is therefore a priority over precision and depending on the ratio of how many times do we consider it more important, we can infer the appropriate limit of n. In particular, with a recall being twice as important, we see that $n = 2$ is the one achieving the highest F score, while from there to $\beta = 10$ we can assess that $n = 3$ is the most appropriate. Of course, the asymptotic behavior favors an unbound value for n because if we wanted to achieve the highest possible recall disregarding precision it would suffice to always return the entire dataset, but this would make the entire process useless.

6.2 Time performances

We will conclude this analysis on the system performance by considering the final throughput achieved by the CPE matcher. As explained in section 2.3, one of the system's requirements is the ability to process the entire asset inventory of potentially multiple clients in a timely manner. Updates in the CPE database are scheduled to happen every 24 hours, which sets a daily deadline for execution.

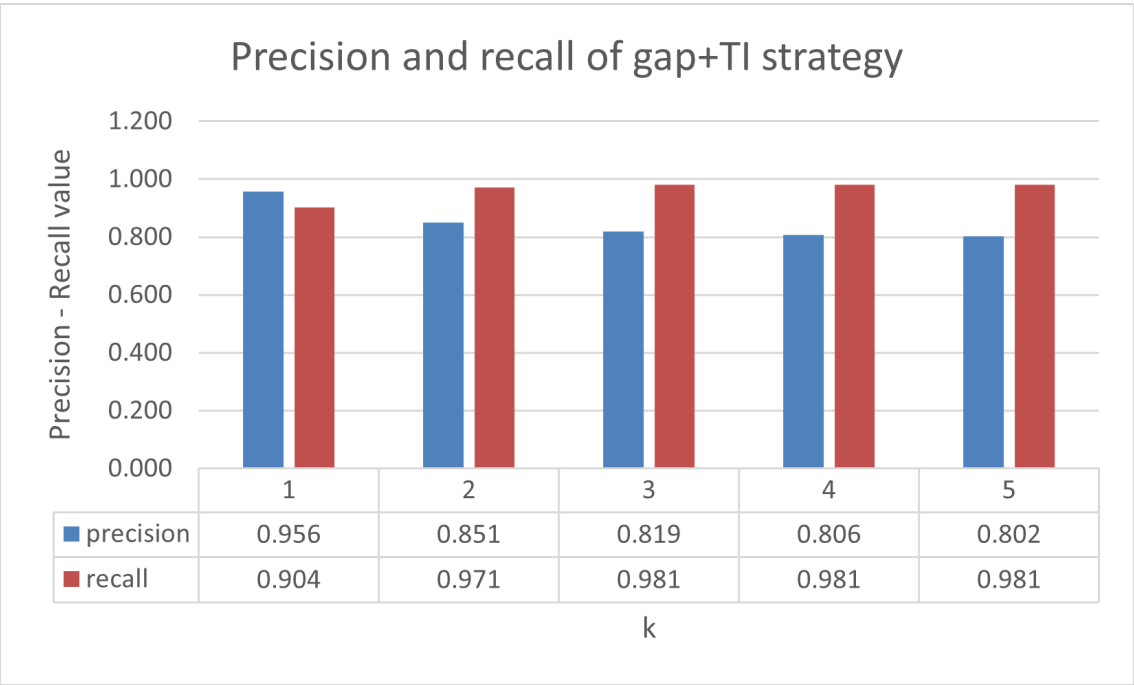
The choice of Go as the main programming language and the use of goroutines, as described in section 5.3, has been done to exploit the parallelizable characteristic of the problem, which allows to process multiple samples and multiple terms in parallel and even out of order.

A graph presenting the effective elapsed time of execution is then presented in Figure 6.5. This graph shows the experienced time of execution for a varying number



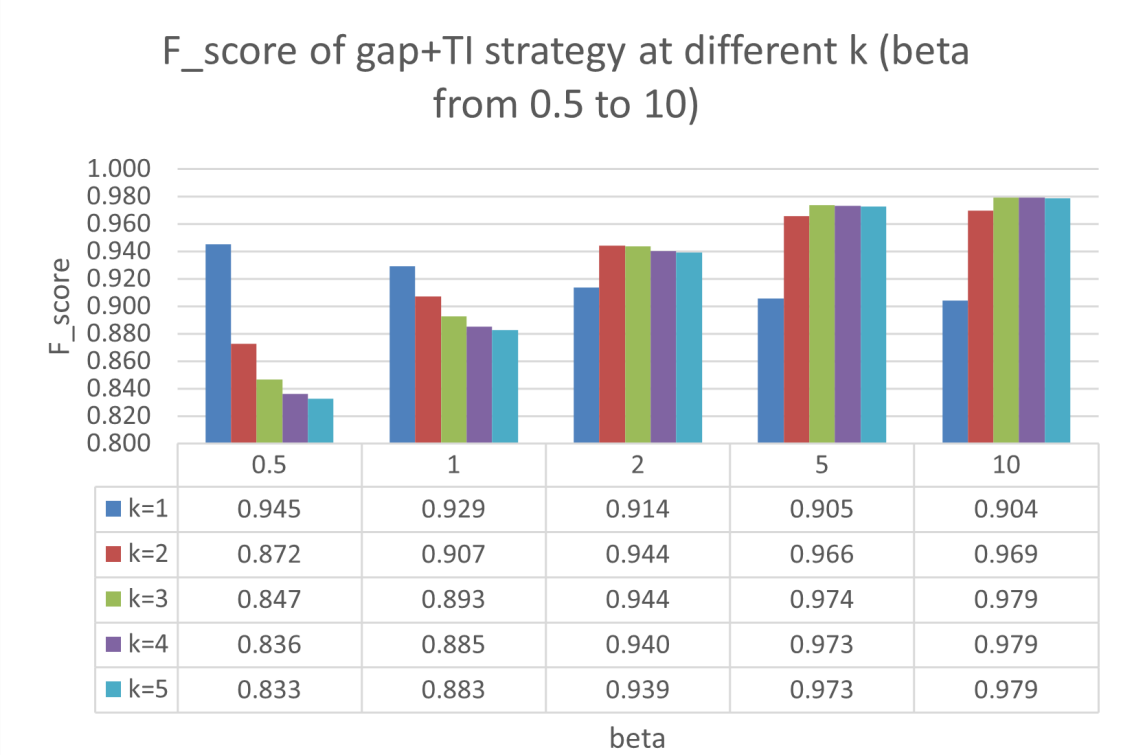
	1	2	3	4	5
default	0.791	0.633	0.523	0.437	0.389
TI	0.929	0.701	0.552	0.458	0.398
gap	0.791	0.810	0.810	0.808	0.809
gap+TI	0.929	0.907	0.893	0.885	0.883

Figure 6.2: F1 score of different strategies (k from 1 to 5)



	1	2	3	4	5
precision	0.956	0.851	0.819	0.806	0.802
recall	0.904	0.971	0.981	0.981	0.981

Figure 6.3: Precision and Recall of the final TI+gap strategy



	0.5	1	2	5	10
k=1	0.945	0.929	0.914	0.905	0.904
k=2	0.872	0.907	0.944	0.966	0.969
k=3	0.847	0.893	0.944	0.974	0.979
k=4	0.836	0.885	0.940	0.973	0.979
k=5	0.833	0.883	0.939	0.973	0.979

Figure 6.4: F_score of the gap+TI strategy at different k and beta

of threads and number of processed samples. Every statistic in this and the following graphs is presented at a 95% confidence interval.

A quick note on the specific number of threads used on the x-axis. The application allows the user to pass as a flag the size of the two thread pools (`CPEWorkerPool` and `TermWorkerPool`). For these experiments, they both were set to the same set of values, 1, 2, 4, 8, 16. Regardless, a single thread is always created to manage the input and output dispatching of samples. Therefore each experiment is composed of effectively 3 $(1+1+1)$, 5 $(2+2+1)$, 9 $(4+4+1)$ threads on so on.

Experiments were done on a AMD Ryzen 7 8845HS chip with 8 cores and 16 logic threads.

We can observe that no matter the number of threads, by computing only 1 sample the elapsed time remains almost constant. This is expected as the application requires an initial warm up time to load the entire CPE dataset into memory and compute the pseudo-posting list using the provided field weights. This process is always required but becomes negligeable once we consider a much larger number of samples.

We can clearly observe the benefits of parallelization which become even more apparent with a large volume of samples to process. This confirms the intuition that the algorithm is primarily cpu-bound. Also, with 1000 samples we can see the system reaching a plateau after 17 threads. We cannot assess with certainty

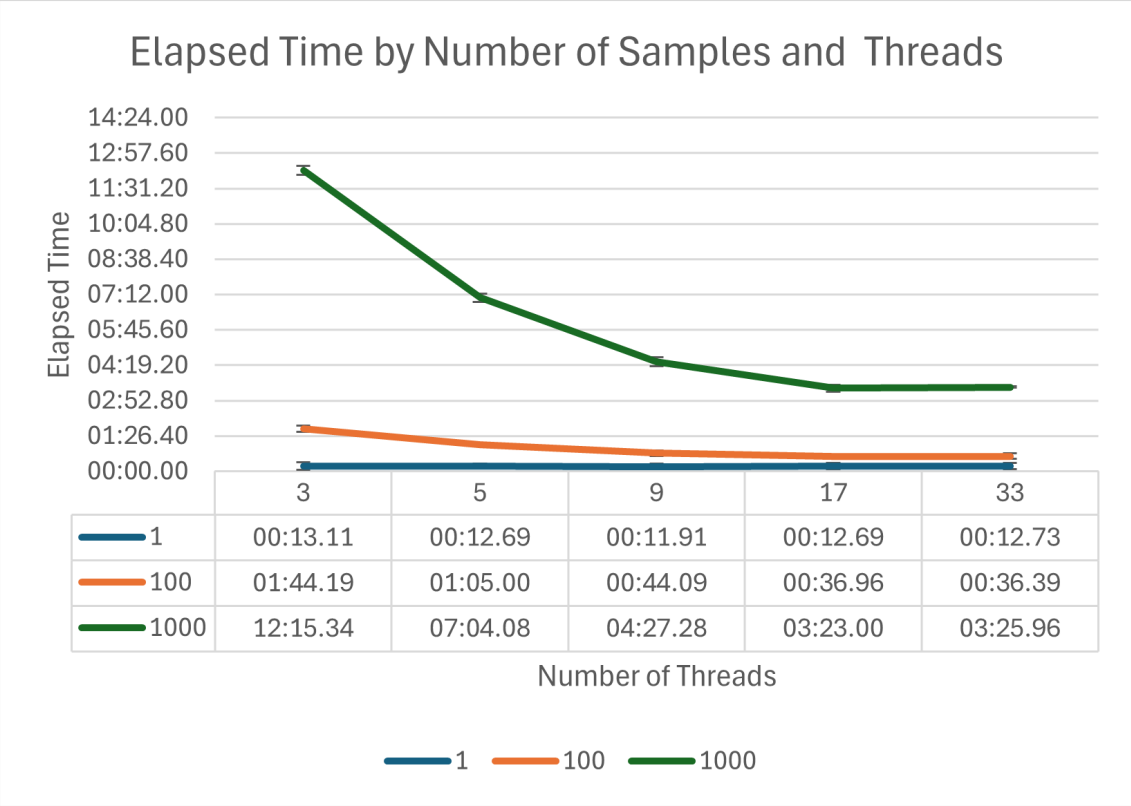


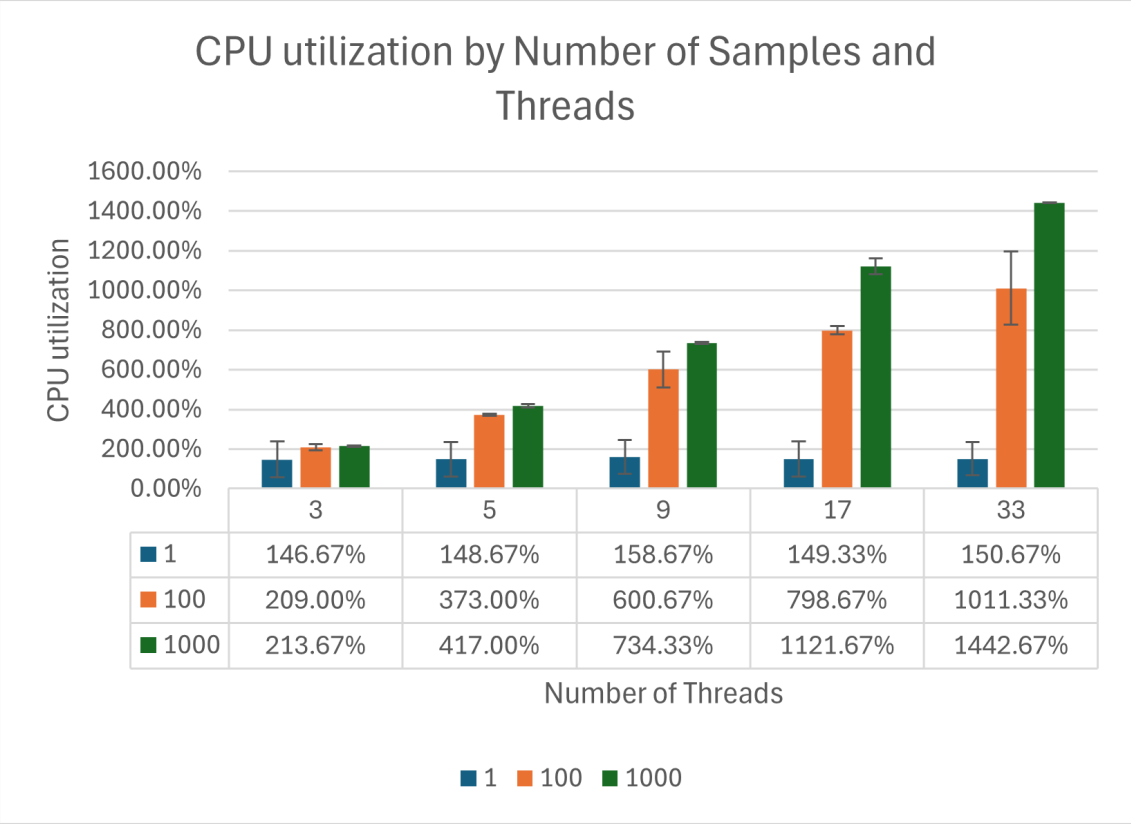
Figure 6.5: Elapsed Time by number of samples and threads

that this behavior is reached because of a hardware limit or some other saturating phenomenon of the program. A more extensive analysis, considering a higher number of available computing units may reveal the true limits of this approach, which should ideally be achieved once the latency due to io operations becomes the limiting factor.

By considering the best achieved result for 1000 samples, obtained with 17 threads, we can estimate the average time required to compute a single sample, $\frac{03:23.00}{1000} = 00 : 00.203$. This in turn allows us to estimate the theoretical maximum number of samples that can be processed in one day as $\frac{24:00:00.000}{00:00:00.203} = 425608.8$. We can then assess that the entire collection of asset inventories should not exceed around 400,000 samples, with the hardware resources that were used for this experiment. Horizontal scalability on the number of computing units may drastically improve those numbers.

Figure 6.6 shows the CPU utilization of those same runs, and indeed we can confirm that the same considerations on improved performance are specifically due to a better usage of the available computing resources. It should be noted however that even with 33 threads in the case of 1000 samples the program still does not reach the theoretical maximum of 1600% utilization and instead peaks at around 1400%.

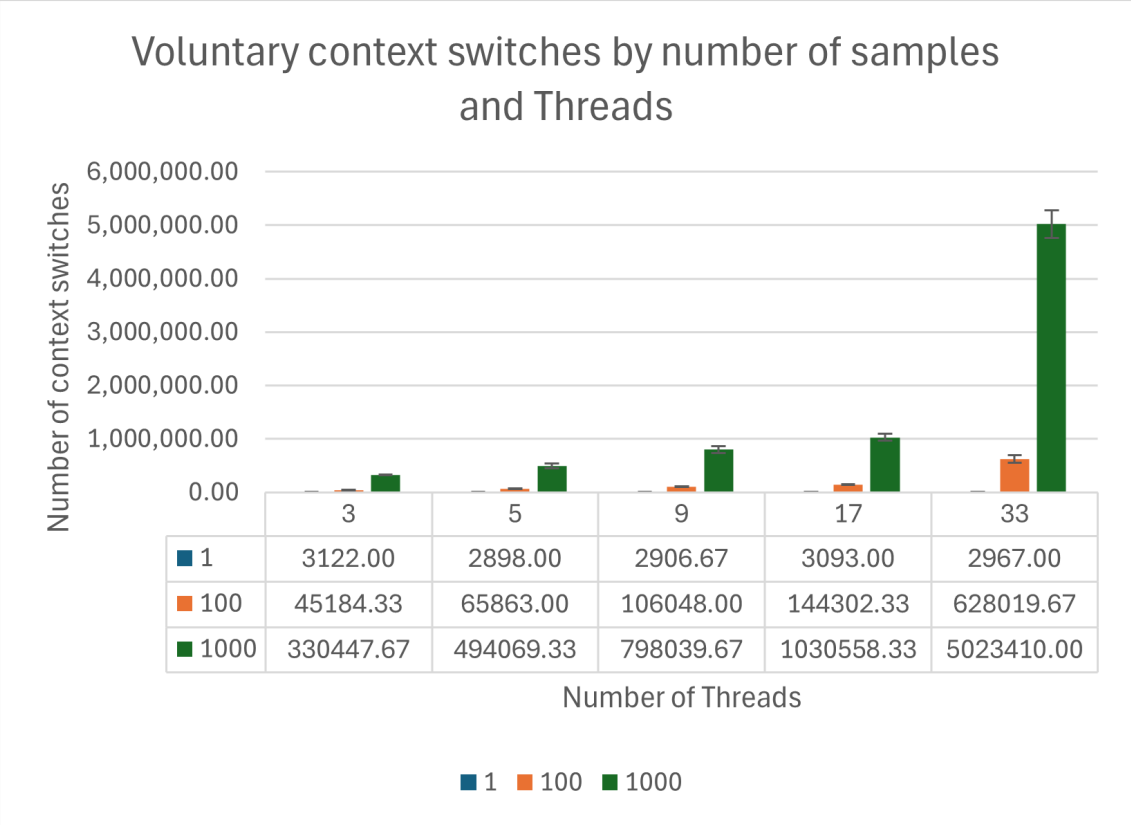
To better understand why we are unable to reach a full utilization of computational resources we can look for the number of context switches that occurred during



	3	5	9	17	33
1	146.67%	148.67%	158.67%	149.33%	150.67%
100	209.00%	373.00%	600.67%	798.67%	1011.33%
1000	213.67%	417.00%	734.33%	1121.67%	1442.67%

Figure 6.6: CPU utilization by number of samples and threads

execution, which are represented in Figure 6.7. The main point of this graph serves to show how the case with 33 threads presents a sudden jump in the number of context switches in comparison to the other cases, with an increase of around 5x the number of switches present in the case with 17 threads. This is another indicator of a very probable saturation of available cores and confirms the nature of this program being heavily cpu-bound.



	3	5	9	17	33
1	3122.00	2898.00	2906.67	3093.00	2967.00
100	45184.33	65863.00	106048.00	144302.33	628019.67
1000	330447.67	494069.33	798039.67	1030558.33	5023410.00

Figure 6.7: Voluntary context switches by number of samples and threads

Chapter 7

Future Developments

More advanced Information Retrieval techniques can be used to improve the core matcher performances. For example, learn to rank, or more sophisticated query expansions methods that could involve the recent progress made in the field of AI, and specifically Language Models may become a valuable resource for this task.

7.1 Software Identification remains an unsolved problem

Software identification is at its core a problem that lies in the way language is used to describe the same thing in different ways. Requirements for an agreed naming method should allow anyone to obtain the same name for the same element and conversely to have only a single name for each element. [23]

7.1.1 Software Identification Ecosystem Option Analysis by CISA

In October 2023, the U.S. Cybersecurity and Infrastructure Security Agency (CISA) published the "Software Identification Ecosystem Option Analysis,"[6], which goes into the complexities surrounding the consistent labeling and tracking of software components. The analysis was motivated by the critical need for organizations of all sizes to effectively manage their software inventory for operational support and, crucially, for vulnerability management. CISA recognized that reliably correlating specific software versions with known vulnerabilities, patches, advisories, and organizational policies is fundamental to cybersecurity, yet this correlation is frequently hampered by inconsistent identification methods.

The primary objective of the CISA analysis is to explore pathways toward an "effective, harmonized software identification ecosystem." Such an ecosystem is considered to overcome the limitations of the current landscape, thereby enabling greater automation in security processes. This would provide clearer visibility into software

inventories, and facilitate the broader and more effective use of Software Bills of Materials (SBOMs).

The document implicitly highlights key challenges inherent in the software identification problem. Foremost among these is the existence of multiple, unharmonized identification solutions, meaning the same software product often carries different identifiers across various tools and databases. This lack of consistency is a major barrier to interoperability and automation. Furthermore, the analysis acknowledges that establishing a unified identification system is not merely a technical challenge but a complex "social task." It requires consensus and agreement across diverse communities on naming conventions and standards, recognizing that there is often no single, objectively "correct" way to identify a software entity.

7.1.2 Intrinsic and Extrinsic tags

Approaches to software identification can be broadly categorized based on the nature of the identifier itself, specifically whether it is intrinsic or extrinsic to the software artifact it represents. Extrinsic identifiers rely on an external registry or authority to maintain the mapping between the identifier and the software object. The validity and meaning of the identifier are dependent on this registry.

Conversely, intrinsic identifiers are derived directly from the software artifact itself or are inherently bound to it, based on an agreed-upon standard or computation method. An example is a cryptographic hash, which acts as a unique digital fingerprint computed from the software's content. Software Heritage identifiers (SWHIDs), which use cryptographic hashes of content, directories, revisions, releases, and snapshots, represent a specific implementation of intrinsic identification for source code artifacts. The key advantage of intrinsic identifiers is their potential for decentralized verification; given the software artifact and its identifier, one can verify the correspondence without consulting a central authority, enhancing trust and resilience. However, it's recognized that both intrinsic and extrinsic identifiers may be needed to satisfy different use cases within the software ecosystem.

Limitations of Common Platform Enumeration (CPE)

For nearly two decades, CPE has been a cornerstone of vulnerability management, enabling organizations to track vulnerabilities and streamline compliance by providing a common language for security tools and databases.

Despite its widespread adoption and historical significance, CPE presents several limitations that challenge its suitability as the sole or primary identifier for the future software ecosystem. A major critique centers on its granularity; CPE was primarily designed to identify broader classes of applications, operating systems, and hardware, making it less effective for the granular identification of specific open-source libraries or software components commonly used in modern development. This lack of specificity becomes problematic when dealing with forked projects, backported security fixes, or software consumed in parts rather than as whole products.

Furthermore, the consistency and coverage of CPE assignments are persistent issues. Software vendors do not always adhere strictly to standardization, leading to inconsistent or non-standard CPE strings for their products. This inconsistency complicates accurate matching against vulnerability databases, sometimes necessitating complex fuzzy matching algorithms and potentially leading to missed vulnerabilities. Additionally, not all software, particularly newer or less common open-source projects, has an official CPE entry in the central dictionary, and maintaining the dictionary's comprehensiveness in the face of rapid software evolution is an ongoing struggle.

These limitations hinder fully automated and reliable vulnerability management. The recognized shortcomings, coupled with the rise of component-based software development and the need for precise identification within software supply chains (as highlighted by SBOM initiatives), are driving the exploration and adoption of alternative or complementary identifiers like Package URLs (PURL), which are often considered more suitable for the package manager ecosystem, and SWID tags. While CPE remains deeply integrated into existing vulnerability management workflows, particularly those reliant on the NVD, its inherent challenges suggest that future-proof software identification ecosystems will likely rely on a combination of identifiers, with approaches like PURL potentially playing an increasingly central role, especially for open-source components.

Package URL

One prominent approach aiming for standardization is the Package URL (purl). A purl provides a specification for a universal, uniform URL string designed to identify and locate software packages across disparate ecosystems, such as Maven, npm, PyPI, NuGet, and others. It standardizes existing conventions into a structured format typically composed of components like scheme (pkg), type, namespace, name, version, qualifiers, and subpath (pkg:type/namespace/name@version?qualifierssubpath). By offering a consistent syntax conforming to established URL standards (RFC 3986), purl facilitates the reliable referencing of software packages in various databases, tools, and APIs, thereby addressing the fragmentation issue inherent in ecosystem-specific naming conventions.

Software Identification Tags

Software Identification (SWID) Tags, defined by the ISO/IEC 19770-2:2015 standard, promise to be an important step towards helping organizations to identify. SWID Tags provide a transparent way for organizations to track the software installed on their managed devices. SWID Tag files contain descriptive information about a specific release of a software product. The SWID standard defines a lifecycle where a SWID Tag is added to an endpoint as part of the software product's installation process and deleted by the product's uninstall process. When this lifecycle is followed, the presence of a given SWID Tag corresponds directly to the presence

of the software product that the Tag describes. The National Institute of Standards and Technology recommends adoption of the SWID Tag standard by software producers, and multiple standards bodies, including the Trusted Computing Group (TCG) and the Internet Engineering Task Force (IETF) utilize SWID Tags in their standards.

NIST plans to continue to promote the incorporation of the SWID Tag standard and associated guidelines in other international consensus standards (such as IETF and TCG efforts), the broad adoption of SWID tagging within the software community, and the use of SWID Tag information in the creation of cybersecurity reference data and security automation content. Additionally, NIST is working to incorporate SWID Tag data into the vulnerability dataset provided by National Vulnerability Database (NVD), and has incorporated use of SWID Tag data into Security Content Automation Protocol (SCAP) version 1.3. These efforts are part of the Software Identification (SWID) Tagging project, which is an initiative of the Computer Security Division's Security Automation Program (SAP). The SAP is focused on standardizing the exchange of security posture information supporting the management of software, vulnerabilities, patches, and secure configurations for computing devices. [30]

7.1.3 Comments

We will summarize key viewpoints and recommendations presented by various stakeholders in response to the CISA Request for Comment (RFC) on its "Software Identification Ecosystem Option Analysis" paper. Each subsection highlights the central arguments of a specific comment, followed by a synthesis of the collective feedback.

Synopsys: Context dependency and Hubrid Models

Synopsys [3] emphasizes that software identification is fundamentally dependent on context, extending beyond simple naming to include version, platform, configuration, and more. They argue that resolving ambiguity often requires understanding this broader context, which even rich taxonomies like CPE 2.3 may not fully capture (e.g., optional configurations). Consequently, Synopsys posits that no single identification scheme can adequately cover all scenarios across diverse development, deployment, and update models. They suggest a contextual, hybrid approach might be necessary, potentially using source code identifiers during development and a CPE-like model for released code. Acknowledging the challenge of private or customized software common in commercial and cyber-physical domains, Synopsys proposes extending the CPE concept to allow for vendor-specific, authoritative dictionaries, potentially hosted by vendors or third parties. Finally, they question the extent to which new identification schemes will improve the practical outcomes of patch management compared to existing methods and solutions.

Google: Multiple Identifiers and Generator Namespacing

Google [1] advocates for embracing the use of multiple software identifiers, recognizing distinct use cases and challenges, particularly differentiating between package manager ecosystems and more decentralized scenarios where central namespaces are lacking. While acknowledging the utility of inherent identifiers like content hashes (e.g., SHA-256) as lookup keys or evidence, Google argues they are impractical as primary identifiers because end consumers typically refer to software by name, vendor, and version. A key recommendation from Google is that namespacing should be applied to identifier generators rather than partitioning the identifier space itself. This approach allows multiple entities (like Google) to create identifiers for their specific builds or patched versions of common software (e.g., a fork of OpenSSL with custom patches) without needing ownership of the identifier space related to the original software, thereby supporting artifact mobility and patch/repackaging scenarios.

GUAC Developers: Identifier Lifecycle and Expanded Properties

The GUAC (Graph for Understanding Artifact Composition) Developers focus on enriching the analysis by introducing the concept of a software identifier lifecycle [4]. They argue that identification should not be viewed as a static, one-time event but as an evolving process involving operations beyond creation, such as discovery, reduction (simplification or mapping), comparison, subtraction, and destruction. This lifecycle perspective is deemed crucial for a more formalized understanding. Furthermore, they propose expanding the set of desirable properties for identifiers beyond those initially listed by CISA. Their suggested properties include associativity (expressing relationships like "part of ecosystem X"), temporality (indicating sequence and distance, relevant for versioning/compatibility), composability (reflecting containment, potentially via Merkle trees like OmniBOR), uniqueness (best achieved via strong cryptographic hashes), agglutination (grouping slight variants, as CPE does for editions/architectures), and human-readability.

Art Manion: Producer Identification as the Key

Art Manion[19], drawing from involvement in SBOM community efforts, proposes a foundational shift in perspective: the key to effective global software identification lies in the unique identification of software producers (developers, vendors, maintainers). Under this model, globally unique producer identities would grant distributed authority and responsibility to these producers for choosing how to identify their own software and communicating this to users. While producers would have freedom in their identification methods (supporting inherent, defined, granular, or grouped identifiers as needed), a minimal central authority might be necessary to manage the top-level namespace for these producer identities, analogous to the domain name system. This approach, Manion suggests, provides a mechanism to connect the currently isolated identification ecosystems (package managers, repositories, proprietary

systems) and aligns conceptually with the "Managed, Distributed Model" (Path 3) outlined in the CISA paper.

Hushmesh: Universal Stem IDs and Automated Remediation

Hushmesh [11] broadens the scope, arguing that the ultimate goal necessitating robust identification is the automation of the full cybersecurity remediation lifecycle at a global scale. This requires a unified namespace not just for software, but for all relevant entities: people, organizations, digital artifacts, and physical devices. They introduce a concept termed "Stem IDs" – identifiers randomly generated from a universal 256-bit namespace that simultaneously function as cryptographic keys. These Stem IDs, they claim, can cryptographically derive numerous keys for signing or encrypting related artifacts, meeting availability, precision, and grouping requirements for various entities. Hushmesh believes a central authority, similar to IANA/DNS root operators, is necessary to manage a registration and lookup service for this system (termed UNS/UCA), distinct from being a centralized data repository. They assert this approach can unify other identifier formats within its framework.

OpenSSF / Linux Foundation: Pragmatism, Use Cases, and International Cooperation

The Open Source Security Foundation (OpenSSF) and Linux Foundation stress the unlikelihood of a single software identifier satisfying all use cases[2]. Instead of striving for one universal standard, they advocate for defining clear conventions on when and how to use which existing identifiers (such as purl, SWHID, CPE) or combinations thereof, tailored to specific real-world problems. The emphasis is on pragmatic application to solve use cases effectively. A crucial point raised is the need for international discussion and cooperation. They urge that solutions be developed collaboratively across governments and industries worldwide, rather than being dictated by any single entity. This global perspective favors solutions less dependent on single central assignment mechanisms (like CPE), preferring internationally scalable approaches such as cryptographic hashes, DNS-based naming, and purl. Ultimately, identifiers should be consistent, scalable, reproducible, and global.

David A. Wheeler: Use Cases, Component Identification, and Related Work

Dr. David A. Wheeler[33], submitting comments as an individual while also affiliated with the OpenSSF and Linux Foundation, strongly advocates for grounding the software identification discussion in specific use cases. He argues that clearly defining what problems software identifiers are intended to solve is crucial before evaluating solutions. As an example, he points to the digital forensics use case of identifying known software files on a system, often addressed using cryptographic hashes via mechanisms like the National Software Reference Library (NSRL) – a relevant identification approach not acknowledged in the CISA paper.

Wheeler also draws attention to the challenge of identifying software components within larger artifacts, particularly in the context of build processes. He mentions OmniBOR as a potentially valuable technology for generating identifiers for source code and derived artifacts even with build variations, but highlights an unresolved challenge: ensuring compatibility between OmniBOR's identification mechanisms and the goal of achieving verified reproducible builds, which are critical for countering certain supply chain attacks. Resolving potential conflicts between these two desirable properties is noted as requiring further work.

Summary

The collective feedback underscores the complexity of the software identification problem, with consensus forming around several key themes. Firstly, there is broad agreement that a single identifier standard is unlikely to meet all needs across diverse ecosystems and use cases. Stakeholders generally advocate for a pragmatic approach involving multiple identifier types (inherent like hashes, defined like purl or CPE) used contextually, potentially in combination, guided by clear conventions. Secondly, the limitations of existing systems, particularly CPE's challenges with granularity, consistency, and coverage for modern software components, are widely acknowledged, prompting calls for alternatives like purl or novel approaches. Thirdly, several responses highlight the need to look beyond static identification, considering the identifier lifecycle (GUAC) or the identity of the producer (Manion) as crucial elements. Fourthly, the scope and ambition vary, from focusing on connecting existing ecosystems (Manion, OpenSSF) to envisioning universal identification across all entities for full automation (Hushmesh). Finally, there is a strong call for international cooperation (OpenSSF) and careful consideration of governance models, ranging from minimal central authorities for producer identity (Manion) or lookup services (Hushmesh) to decentralized approaches favored for global scalability (OpenSSF). The challenge is recognized as both technical and social, requiring community agreement on standards and practices.

7.2 Alternatives approaches

In the meantime, there can be alternative approaches to just CVE filtering. For example, machine learning techniques can be employed to inspect new vulnerability reports and somehow find the relevant vulnerably software in an inventory.

Generative architectures, on the other hand, do not seem to be appropriate for this task, unless it is guaranteed that the same identification is always generated for the same software.

Efforts such as secureBERT [8] might be useful in recognizing relevant segments inside a vulnerability report, which would help identify the affected software.

Zhang et al. [34] propose a machine learning-based approach for automated vulnerability remediation analysis. The proposed decision tree based framework has

only been tested on electric utilities but can be applied to many other organizations, especially critical infrastructures. Results showed high prediction accuracy and time savings and demonstrate the value in applying machine learning to automate remediation processes.

Waltermire [31] from NIST presents some guidelines for the creation of interoperable software identification (SWID) tags and their usage as part of a comprehensive software lifecycle.

The 'SBOM Forum' [5] has developed a proposal, which describes a small set of steps that can be taken to alleviate a large portion of the naming problem. Their solution leverages intrinsic and native identifiers. For software identification, the solution is based on Package URLs, which, unlike CPEs, are intrinsic identifiers deerived from a software product's attributes and do not require a central registry.

To address the challenge of matching software inventory to vulnerability reports, Huff et al.[13] propose a recommender system. This system leverages a pipeline of natural language processing (NLP), fuzzy matching, and machine learning techniques to automatically identify a minimal set of candidate Common Product Enumerators (CPEs) for software names. By doing so, the system aims to enhance the accuracy of vulnerability identification and alerting, significantly reducing the manual effort typically required by cybersecurity analysts for software product vulnerability matching

Ushakov et al. [29] propose an algorithm for mapping the software products names in the analyzed system logs to the relevant Common Platform Enumeration entries in open vulnerability databases based on the Ratcliff/Obershelp algorithm, identification of known vulnerabilities for the detected entries, and security risk assessment of the analysed system.

Hu et al. [12] propose a CPE-Identifier system, an automated CPE annotating and extracting system, from the CVE summaries. The system can be used as a tool to identify CPE entities from new CVE text inputs.

The work of Zhang et al. [35] studies dynamic risk-aware patch scheduling to determine the order of patching vulnerabilities and minimize the security risk brought by vulnerabilities.

Joshi et al. [16] demonstrate a prototype for an entity and concept spotting framework that identifies cybersecurity-related concepts from heterogeneous data sources, which aligns and links them to relevant resources on the Web using the IDS ontology[1].

Tovarnak et al.[28] propose a graph-based approach for identification of vulnerable asset configurations via CPE matching. The approach consists of a graph model and insertion procedure that is able to represent and store information about CVE vulnerabilities and also about hierarchies of asset components classified by CPE names.

[1]http://ebiquity.umbc.edu/IDSv2.0.1.owl

Chapter 8

Acknowledgements

Voglio ringraziare i miei professori per l'aiuto fornito durante la stesura di questa tesi e tutti i ragazzi del consorzio Metis per l'accoglienza ed il supporto durante questi mesi.

Sono infinitamente grato alla mia famiglia, in particolare ai miei fratelli e ai miei genitori per il costante sostegno durante tutto il mio percorso di studi e di crescita, oltre che per avermi sempre permesso di inseguire le mie passioni.

Ringrazio tutti i miei amici e compagni di corso, con i quali ho condiviso questi splendidi anni universitari. Fabio e Giacomo per la sintonia e la complicità che ci ha permesso di svolgere tutti i nostri progetti, tutti i ragazzi la cui amicizia dura anche sin dalla triennale, tra cui Antonino, Davide, Elia, Francesco, Massimiliano e Mirco, ed il gruppo dei lucchesi Alice, Antonio e Fabio.

Un ringraziamento speciale a Gabriele, a cui sarò sempre grato per avermi insegnato a pensare.

Bibliography

[1] Comment Submitted by Google LLC.

[2] Comment Submitted by OpenSSF.

[3] Comment Submitted by Synopsys Inc.

[4] Comment Submitted by The GUAC Project.

[5] A Proposal to Operationalize Component Identification for Vulnerability Management.

[6] Software Identification Ecosystem Option Analysis.

[7] 2024_open_source_security_and_risk_analysis_report_wrapped, 2024.

[8] Ehsan Aghaei, Xi Niu, Waseem Shadid, and Ehab Al-Shaer. SecureBERT: A Domain-Specific Language Model for Cybersecurity, October 2022. arXiv:2204.02685 [cs].

[9] Brant A Cheikes, David Waltermire, and Karen Scarfone. Common platform enumeration : naming specification version 2.3. Technical Report NIST IR 7695, National Institute of Standards and Technology, Gaithersburg, MD, 2011. Edition: 0.

[10] Paul Cichonski, David Waltermire, and Karen Scarfone. Common platform enumeration : dictionary specification version 2.3. Technical Report NIST IR 7697, National Institute of Standards and Technology, Gaithersburg, MD, 2011. Edition: 0.

[11] Manu Fontaine. Comment Submitted by Hushmesh Inc.

[12] Wanyu Hu and Vrizlynn L. L. Thing. CPE-Identifier: Automated CPE identification and CVE summaries annotation with Deep Learning and NLP, May 2024. arXiv:2405.13568 [cs].

[13] Philip Huff, Kylie McClanahan, Thao Le, and Qinghua Li. A Recommender System for Tracking Vulnerabilities. In *Proceedings of the 16th International Conference on Availability, Reliability and Security*, ARES '21, pages 1–7, New York, NY, USA, August 2021. Association for Computing Machinery.

[14] ITRC. 2023-Annual-Data-Breach-Report, January 2024.

[15] Yuning Jiang, Manfred Jeusfeld, and Jianguo Ding. Evaluating the Data Inconsistency of Open-Source Vulnerability Repositories. In *Proceedings of the 16th International Conference on Availability, Reliability and Security*, ARES '21, pages 1–10, New York, NY, USA, August 2021. Association for Computing Machinery.

[16] Arnav Joshi, Ravendar Lal, Tim Finin, and Anupam Joshi. Extracting Cybersecurity Related Linked Data from Text. In *2013 IEEE Seventh International Conference on Semantic Computing*, pages 252–259, September 2013.

[17] Research Juniper. vulnerable-software-supply-chains-are-a-multi-billion-dollar-problem, May 2023.

[18] Tuomas Ketola and Thomas Roelleke. BM25-FIC: Information Content-based Field Weighting for BM25F.

[19] Art Manion. Comment Submitted by Art Manion.

[20] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[21] Charlie Osborne. Software Supply Chain Report 2023.

[22] Mary C Parmelee, Harold Booth, David Waltermire, and Karen Scarfone. Common Platform Enumeration: Name Matching Specification Version 2.3. Technical Report NIST IR 7696, National Institute of Standards and Technology, NIST IR 7696, 2011.

[23] Ruben Ramirez. Software Identification Challenges and Guidance.

[24] Stephen Robertson and Hugo Zaragoza. On rank-based effectiveness measures and optimization. *Information Retrieval*, 10(3):321–339, June 2007.

[25] Stephen Robertson and Hugo Zaragoza. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval*, 3:333–389, January 2009.

[26] Stephen Robertson, Hugo Zaragoza, and Michael Taylor. Simple BM25 extension to multiple weighted fields. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 42–49, Washington D.C. USA, November 2004. ACM.

[27] Karen Sparck Jones. A STATISTICAL INTERPRETATION OF TERM SPECIFICITY AND ITS APPLICATION IN RETRIEVAL. *Journal of Documentation*, 28(1):11–21, January 1972.

[28] Daniel Tovarňák, Lukáš Sadlek, and Pavel Čeleda. Graph-Based CPE Matching for Identification of Vulnerable Asset Configurations. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 986–991, May 2021. ISSN: 1573-0077.

[29] Roman Ushakov, Elena Doynikova, Evgenia Novikova, and Igor Kotenko. CPE and CVE based Technique for Software Security Risk Assessment. In *2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, volume 1, pages 353–356, September 2021. ISSN: 2770-4254.

[30] David Waltermire and Brant Cheikes. Forming Common Platform Enumeration (CPE) Names from Software Identification (SWID) Tags. Technical Report NIST IR 8085, National Institute of Standards and Technology, NIST IR 8085, 2015.

[31] David Waltermire, Brant Cheikes, Larry Feldman, and Gregory Witte. Guidelines for the Creation of Interoperable Software Identification (SWID) Tags. Technical Report NIST IR 8060, National Institute of Standards and Technology, NIST IR 8060, 2016.

[32] David Waltermire, Paul Cichonski, and Karen Scarfone. Common platform enumeration : applicability language specification version 2.3. Technical Report NIST IR 7698, National Institute of Standards and Technology, Gaithersburg, MD, 2011. Edition: 0.

[33] Dr David A Wheeler. Comments to the CISA Request for Comment (RFC) on the Software Identification Ecosystem Analysis White Paper.

[34] Fengli Zhang, Philip Huff, Kylie McClanahan, and Qinghua Li. A Machine Learning-based Approach for Automated Vulnerability Remediation Analysis. In *2020 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9, June 2020.

[35] Fengli Zhang and Qinghua Li. Dynamic Risk-Aware Patch Scheduling. In *2020 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9, June 2020.