b
UNIVERSITÄT
BERN

An Investigation into Vulnerability Databases

Bachelor Thesis

Brian Schweigler
from
Bümpliz BE, Switzerland

Faculty of Science
University of Bern

31 May 2020

Prof. Dr. Oscar Nierstrasz
Pascal Gadient

Software Composition Group
Institute of Computer Science
University of Bern, Switzerland

Abstract

The vulnerability databases' affiliations and contributions are non-trivial and have not yet been studied in depth. This raises a major concern regarding the correctness of the data used in numerous existing studies. To investigate this problem, we first collected publicly available data from the websites of five major database providers, and then we normalized and correlated the individual entries to track them within different vulnerability databases.

370,298 security reports were extracted, 89% of which were accessible at more than one provider. Surprisingly, many reports were inconsistent with respect to scores and detail descriptions. In the scoring system CVSS version 3.0, for example, we found on average a difference of 0.8 on *NVD* and *Snyk*, whereas CVSS version 2.0 remains largely consistent with a difference of only 0.1 between *NVD* and *RAPID7*.

Furthermore, we discovered that the security-related popularity differs for widely used software, and we show that shared code bases but not library usages can be predicted by aggregating security reports over periods of time. Finally in visualizations, software release cycles become visible.

Contents

1 Introduction **1**

2 Background **4**
2.1 Reporting 4
2.2 Publication 4
2.3 Severity Scoring 5

3 Vulnerability Databases **7**
3.1 Used Vulnerability Database Providers 7
3.2 Features 8

4 Empirical Study **10**
4.1 Methodology 10
4.2 Features 11
4.3 Propagation among Databases 12
4.4 Scoring 13
4.5 Trends 15
4.5.1 Shared Code 15
4.5.2 Shared Libraries 16
4.5.3 Related Software 18

5 Outlook **20**

6 Threats to Validity **22**
6.1 Completeness 22
6.2 Correctness 22

7 Related Work **24**

8 Conclusion **26**

A Anleitung zum wissenschaftlichen Arbeiten **31**

1 Introduction

A vulnerability database is a private or public platform that collects, maintains, and disseminates data about software-related security implications. Vulnerability databases aggregate knowledge through so called "security reports", where each report describes properties of a single vulnerability. In general, a report mentions one or more uniquely assigned ids, the affected product, a brief description of the problem, relevant dates, and optionally the authorship, instructions on how to reproduce the problem, workarounds or updates remedying the problem, sample code, an impact score, external references, and special remarks. The major impact score, *i.e.*, a risk assessment metric, used in security reports is the *Common Vulnerability Scoring System (CVSS)* [27].

Vulnerability databases are a valuable resource for software developers to efficiently discover and fix issues in their code. Despite the risk of misuse [2, 14], this data also fosters reasoning among security researchers and developers, *e.g.*, to discover trends that might require attention. In this regard, much work that relies on vulnerability database information has been published, however, either the integrity of the data was not much of a concern, or only a single database was considered. For example, there have been attempts at coalescing security reports from multiple databases, but none of them have verified the integrity of the data [1, 26, 23, 15]. Furthermore, Chen *et al.* verified security reports and predicted invalid entries, but only for one database [3]. Similarly, CVSS scores received much attention [4, 8], but their divergence across different databases has not yet been considered.

This work represents a first step towards understanding the usage, the maintenance, the spread, and the integrity of such data. Moreover, we try to leverage benefits enabled by the large amount of data we collected, *e.g.*, to find correlations between different products. This information can pave the way for more accurate and efficient vulnerability analysis or software patching which ultimately leads to increased security. Henceforth, we focus on the following three research questions:

RQ$_1$: *What data is collected, and how is the information propagated between the different vulnerability databases?*
We downloaded and processed 370 298 security reports from five major vulnerability databases. We collected and compared the features available in the different databases, and we searched for duplicated data. Based on the found duplicates, we reconstructed a graph that reveals the information propagated between the databases. We encountered up to eleven different features in a single database, but only unique ids, brief descriptions, relevant dates and references to other vulnerabilities were commonly used. We realized that *CVE* and *NVD* are sharing data with every other database, whereas *RAPID7*, *Exploit-DB*, and *Snyk* only share data with selected competitors. We conclude that, in every database except CVE and NVD, most of the data is originating from at least one other provider, frequently from CVE or NVD.

RQ$_2$: *Are vulnerability scores used consistently?*
We matched security reports across different databases and extracted the found scores. We realized that different metrics are in use, *i.e.*, CVSS version 2.0 (CVSSv2), CVSS version 3.0 (CVSSv3), and CVSS version 3.1 (CVSSv3.1). Moreover, depending on the databases we either found only minor differences that seem to emerge from rounding errors (CVSSv2, NVD and RAPID7), or rather large changes, *i.e.*, up to 12% on average, that indicate a different valuation scheme (CVSSv3, NVD, and Snyk).

RQ$_3$: *Can we predict trends based on the collected data?*
We extracted the 33 most frequent words from descriptions and titles that relate to a product or brand name. For each of these words, we collected all security reports that contained the word at least once, and their publication date. With this information we generated one plot for each keyword where we can inspect the report submission frequency for a specific software product over several years. When we overlay certain plots, we can clearly see that some suffer from security problems at the same time, *e.g.*, caused by a shared code base (Firefox/Thunderbird). However, the security reports for an application and a used library, *e.g.*, FFmpeg/Android, do not correlate. In addition, we can derive the patch or release cycles based on the frequency of vulnerability reports.

The remainder of this thesis is organized as follows: We provide the required background to understand certain terminology used throughout the thesis in chapter 2, elaborate on the state of the art in vulnerability databases in chapter 3, and present the results of our empirical study in chapter 4. Finally, we provide an outlook on how

vulnerability databases could be improved in chapter 5, recap the threats to validity in chapter 6, and summarize related work in chapter 7. We conclude in chapter 8.

2

Background

In this chapter, we briefly explain how vulnerability reporting works to facilitate the understanding of problems associated with existing processes that we mention in the subsequent chapters of this thesis.

2.1 Reporting

Individuals, institutions and companies that encounter a bug reason about its severity, and if the impact is not negligible, they file a security report with a vulnerability database provider. The standard procedure is to use a submission form or email template. After submission, the provider will either reply with further inquiries, ask for a revision if required details are not provided as requested, or accept the submission.

2.2 Publication

If a published submission is accepted, a *CVE Numbering Authority (CNA)* will assign a unique *CVE-ID* following the structure *CVE-YYYY-NUMBER*, *e.g.*, CVE-2020-1342, that can be used as a shared identifier for referencing a vulnerability across the internet. The CNA is usually the *Mitre Corporation* which is the operator of the CVE database, but there exist various other CNAs, *e.g.*, maintained by Microsoft, Red Hat and other large companies. Next, the potential impact is assessed by several experts who then

collectively decide on a severity score for the security report. Prior to the publication, there is a grace period between the acceptance of a submission and the publication of the report to allow affected software vendors to incorporate patches, before attacks can be leveraged. The grace period depends on several factors, *e.g.*, severity, complexity, cooperation, and internal policies.

2.3 Severity Scoring

The *Common Vulnerability Scoring System (CVSS)* is a defacto standard used in security reports and generally considered a trustworthy and robust scoring system for vulnerabilities [8]. According to FIRST (Forum of Incident Response and Security Teams), the "CVSS provides a way to capture the principal characteristics of a vulnerability and produces a numerical score reflecting its severity" [16]. A CVSS score is compiled from multiple attributes, *i.e.*, the version-specific "CVSS Breakdown Vectors", each covering a different security-related aspect.

There exist four different versions of the CVSS specification. CVSS version 1.0 (CVSSv1) was not subjected to mass peer review across multiple organizations or industries.[1] Hence, only subsequent CVSS standards are in use, namely version 2.0 (CVSSv2), version 3.0 (CVSSv3) and the newly released version 3.1 (CVSSv3.1). Compared to version 1.0, version 2.0 has an improved scoring algorithm that more accurately reflects the actual threat. The major differences between version 2.0 and 3.0 are the inclusion of the breakdown vectors "Scope" and "User Interaction", and the renaming of old labels, *e.g.*, "Authentication" changed to "Privileges Required". In the recent version 3.1 update, no new metrics or metric values were introduced, instead, various minor optimizations have been applied to the specification. For instance, the formulas used to calculate base, temporal and environmental scores were altered, as well as floating-point number rounding.[2] CVSSv1 was released on APR-2005,[3] version 2.0 on JUN-2007,[4] version 3.0 on JUN-2015,[5] and version 3.1 on JUN-2019.[6]

CVSS scores range from zero to ten regardless of the version. The severity of CVSSv2 scores is ranked in low (0.0-3.9), medium (4.0-6.9), and high (7.0-10.0) tiers, whereas CVSSv3 specifies the tiers none (0.0), low (0.1-3.9), medium (4.0-6.9), high (7.0-8.9), and critical (9.0-10.0). The higher a vulnerability is ranked the more dangerous it is in the wild. A critical vulnerability can usually be triggered remotely without any user intervention, is wide spread, and enables full control over the remote computer. For example, the *Quram qmg* image library contained a buffer overwrite vul-

[1] `https://www.first.org/cvss/v2/history`
[2] `https://www.first.org/cvss/v3.1/specification-document`
[3] `https://www.first.org/cvss/v1/`
[4] `https://www.first.org/cvss/v2/`
[5] `https://www.first.org/cvss/v3-0/`
[6] `https://nvd.nist.gov/General/News/CVSS-v3-1-Official-Support/`

nerability that enabled arbitrary remote code execution without any user interaction.[7] This vulnerability could be exploited through a specially crafted MMS message and was present in all Samsung Android-powered devices starting with Android 8.

[7] https://nvd.nist.gov/vuln/detail/CVE-2020-8899

3

Vulnerability Databases

In this chapter, we present the used vulnerability databases, and we discuss their features.

3.1 Used Vulnerability Database Providers

The five database providers we selected are widely used based on the search-engine popularity, their activity, and the recency of entries. In Table 3.1 we show detailed statistics about each database.

Database	Owner	Founded	Reporters	Business model
CVE	Mitre	1999	everybody	not-for-profit corporation
NVD	NIST	2000	members	not-for-profit agency
Exploit-DB	Offensive Security	2004	everybody	for-profit corporation
RAPID7	RAPID7	2000	members	for-profit corporation
Snyk	Snyk Ltd.	2006	members	for-profit corporation

Database	Special features	First entry	Cut-off date	Database size
CVE	data download, comprehensiveness	30-DEC-1999	12-DEC-2019	156 828
NVD	data download, CVSS scores	01-OCT-1988	03-DEC-2019	133 477
Exploit-DB	exploit downloads	23-MAR-2003	01-NOV-2019	44 539
RAPID7	references to Metasploit	02-NOV-1988	18-DEC-2019	30 849
Snyk	-	20-MAY-2007	17-NOV-2019	4 605

Table 3.1: Overview of vulnerability databases used in this work

The *Common Vulnerabilities and Exploits (CVE)* database [20] provides the CVE-IDs that are assigned to vulnerabilities and exploits which can be found in other vulnerability databases and across the internet. CVE only provides incomplete downloads of their data feed archive. Therefore, we still had to manually search and scrape their entire website to gather the missing data. They do not provide any vulnerability scores. The *National Vulnerability and Exploit (NVD)* database [22] is owned by the state-controlled *National Institute of Standards and Technology (NIST)*, under the *US Department of Commerce*. Generally, *NVD* extends the *CVE* database with CVSS vulnerability scores and provides better structured data, *e.g.*, the affected software configurations are reported separately in contrast to *CVE*'s listing at the end of the description. The *Exploit-DB* database [17] has a very varied history: it started as a public archive under the name "FrSIRT" in 2004, changed to "VUPEN" in 2008 and handed off to the for-profit company Offensive-Security in 2009, under the current name. They now use the content in the database for their offered security courses. Unlike the others, *Exploit-DB* usually provides exploits and direct links to affected software. The *RAPID7* database [18] is maintained by a for-profit company that generates revenue with corresponding code assessment frameworks and consulting services. *RAPID7* owns Metasploit, a leading open-source penetration testing framework. The *Snyk* database [19] is maintained by a for-profit company that provides code assessment tools, which are free for open-source projects.

Every provider curates vulnerabilities for closed and open-source applications. We only used publicly available information, however not all information was always accessible, *e.g.*, *RAPID7* provides more detailed vulnerability scores when using one of their paid plans. The database with the longest history is *NVD*, and the one with the shortest is *Snyk*. However, an early first entry or founding year do not indicate a large database. For example, *Exploit-DB* has its first entry from 2003, but still has more entries than *RAPID7* whose first entry dates back to 1988. Similarly, *Exploit-DB* was founded four years after *RAPID7*, but its size is still larger.

Only *NVD* and *Exploit-DB* allow everybody to file a security report, while for the other providers a membership is required. The date of the most recent entry included in our experimental study depends on the database's update cycle, but is within 01-NOV-2019 and 18-DEC-2019.

3.2 Features

Each database has varying features, but some overlap between the different vulnerability databases. An overview of the feature support is shown in Table 3.2, which we will discuss in more detail in section 4.2 and in section 4.3. In particular, we identified

Database	**CVE-ID**	**CWE**	**Authorship**	**Title**	**Description**	**References**	**Dates**	**Scores**
CVE	✔	✖	(✔)	(✔)	✔	✔	✔	✖
NVD	✔	✔	(✔)	(✔)	✔	✔	✔	✔
Exploit-DB	✔	✖	✔	✔	(✔)	(✔)	✔	✖
RAPID7	✔	✖	✖	✔	✔	✔	✔	✔
Snyk	✔	✔	✔	(✔)	✔	✔	✔	✔

Table 3.2: Feature support of vulnerability databases

these common features:

i) *CVE-ID:* a unique identifier provided by the *CVE*, ii) *CWE:* one or more entries from the *Common Weakness Enumeration (CWE)* that is a formal list of software weakness categories to provide a common baseline standard for weakness types [21], iii) *Authorship:* information pertaining to authors of a security report, iv) *Title:* a unique title, usually made up of the CVE-ID (if assigned), as well as affected software configurations and or platform, v) *Description:* additional information regarding the vulnerability, often including possible causes or effects, further referring to guidelines and materials on how to exploit a vulnerability, vi) *Reference:* reference(s) to other related entries, additional information, or other vulnerability databases, vii) *Date:* the submission, publication, or modification date(s), and finally, viii) *Scoring:* the vulnerability scoring according to CVSSv2 or CVSSv3.

4
Empirical Study

Besides the used methodology, we discuss in this chapter the gained insights from the gathered data. In particular, we focus on the use of features, the propagation of data among databases, the scoring, and trends. In other words, we try to answer $\mathbf{RQ}_1$ in section 4.2 and section 4.3, $\mathbf{RQ}_2$ in section 4.4, and finally, $\mathbf{RQ}_3$ in section 4.5.

4.1 Methodology

We acquired all data by scraping the publicly accessible websites of the vulnerability databases using regular expressions, except for *CVE* and *NVD* which both provide curated JSON data feed downloads. Next, we normalized the data, *i.e.*, we fixed invalid escape character sequences, manually verified entries with a description of more than 1 500 characters for correctness, and ensured that identical id types are following the same structure. Ultimately, we collected 370 298 entries and stored that data in a relational SQLite database, each database provider having their own distinct table, with column names matching the naming used by database providers as closely as possible. Despite changes between CVSSv3 and CVSSv3.1 being small, we ensured that all data collected on *NVD* is based on the 3.0 standard, instead of the 3.1 standard, verified through the breakdown vector that specifies the version used among its attributes. On *Snyk* this was not possible, as 115 out of 4 605 entries already used the CVSSv3.1 at the time of indexing, with no means to access entries prior to the update reliably.

For the trends, we extracted the titles and descriptions of all security reports except those from *CVE*, and then we determined the frequencies of all used words. We excluded *CVE*, because more than 90% of the content was identical to *NVD*; the differences were only caused by reserved CVE-IDs unavailable in *NVD*. Next, we sorted the word frequencies of that list in descending order before we removed stop-words by using the Python `stop-words` package.[1] We used the package's built-in stop-word set for the English language and expanded it with custom words after manual review of the preliminary list. Subsequently, we reviewed the 200 most frequent words, where we manually separated those relating to a product or brand name. This process led to 33 product-oriented keywords. For the next step, we categorized all security reports according to the product-oriented keyword occurrences or excluded them if no keyword matched. For every categorized security report we collected the provided publication date. With this information we can assess keywords in relation to their usage over time.

The complete SQLite database, the used Kotlin and Python scripts as well as the adapted stop-word list are available in the public Github repository.[2]

4.2 Features

In this and the subsequent section we try to answer the question *What data is collected, and how is the information propagated between the different vulnerability databases?* Table 3.2 illustrates the feature support of the different vulnerability databases.

We found that 89% of all collected entries contain a unique identifier based on the CVE-ID. In addition, *Exploit-DB*, *RAPID7*, and *Snyk* assign custom IDs only valid within their own ecosystems. A CWE categorization is only available in *NVD* and *Snyk*. Only *Snyk* and *Exploit-DB* share the author(s) of published reports, which may be organizations, institutions, or individuals. *NVD* and *CVE* provide authorship information, however *NVD* indicates itself as an assigner on all its entries, with no further information on who submitted the corresponding report, and *CVE* frequently provides the author "Mitre", *i.e.*, the company behind *CVE*. A descriptive title is available for entries published in *Exploit-DB*, *RAPID7*, and *Snyk*, whereas *CVE* and *NVD* simply use the CVE-ID. A description is available from all providers, however *Exploit-DB*'s description includes instructions for leveraging the vulnerability, and if available, the exploit. References to other related entries, additional information, or other vulnerability databases are provided in all databases. However, *Exploit-DB* officially only supports references to *CVE* and *NVD*. Additional external references must be stated within descriptions which do not follow a predefined structure and thus are hard to parse. Every vulnerability database provider dates their reports, but all of them use different terms: *CVE* provides the "Date Entry Created", *NVD* the "Published Date",

[1] `https://pypi.org/project/stop-words/`
[2] `https://github.com/Brian6330/VulnerabilityDBInvestigations`

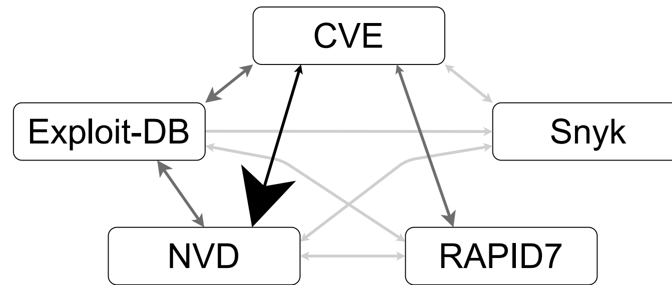

Figure 4.1: Information propagated between database providers

Exploit-DB the “Date”, *RAPID7* the “Date Added”, and *Snyk* the “Disclosure Date”. Moreover, *NVD* and *Snyk* also include the date of the last modification.

Only three databases maintain risk scores: *NVD* supports CVSSv2 and CVSSv3, *RAPID7* only version 2, and *Snyk* exclusively version 3. We further encountered various other features specific to a database: For example, *CVE* provides voting information and comments for older entries. *EXPLOIT-DB* provides attacks including exploits and a direct link to the affected app, if publicly accessible. *NVD* lists detailed information on affected software configurations and a history of changes. *RAPID7* links to related vulnerabilities. *Snyk* includes Proof-of-Concepts (PoC) and remediation strategies.

Many of the provided features are interpreted differently for each database. This introduces burdens for security researchers that have to manually disassemble the provided information.

4.3 Propagation among Databases

Considering the features and their values, *e.g.*, references to other databases, we discovered the relationships presented in Figure 4.1. The arrows represent the information flow based on the found references. The larger an arrowhead is, the more information has been propagated. The arrow color illustrates the number of involved entries, *i.e.*, black for more than 100 000 entries, dark grey for more than 5 000 entries, and light grey for the remaining entries. All databases inherently link to *CVE*. It seems that the information available at *CVE* is the baseline for all other providers, which is augmented with a vulnerability score by *NVD*. This information is then (partially) reused by every provider. Interestingly, *EXPLOIT-DB* has references to *RAPID7*, in the form of entries submitted by *RAPID7*, but none are submitted by *SNYK*, even though both *RAPID7* and *SNYK* reference *Exploit-DB*. Since all commercial databases refer at most to one other commercial database, and they seldomly reference each other with hyperlink-references, we believe that this is on purpose due to competition.

The origin of information is not reported transparently for the majority of databases. We would like to see more transparency to increase the verifiability.

4.4 Scoring

In this section we try to answer the question *Are vulnerability scores used consistently?*

The CVSS is used among the three database providers that provide scoring, *i.e.*, *NVD*, *RAPID7*, and *Snyk*. *NVD* is the only one that uses both the CVSSv2 and CVSSv3.1. *RAPID7* only uses integers, instead of numbers with a single decimal digit and does not specify the CVSS version. We assume that version 2 is used, as the characteristics of the break-down vector fit CVSSv2. *Snyk* uses CVSSv3.1 on newer entries, particularly those after the release of CVSSv3.1, while retaining the version 3.0 rating on older entries.

We inspected entries that exist across databases, based on matching CVE-IDs, which results in 18 722 CVSSv2 entries available in *NVD* and *RAPID7* and 3 201 CVSSv3 entries available in *NVD* and *Snyk*. Considering the average CVSSv2 security report score, *NVD* and *RAPID7* have almost the same scores, 6.2 and 6.3 respectively, with the difference of 0.1 likely traced back to *RAPID7* using integers instead of numbers with a single decimal digit. The average CVSSv3 score is 7.4 on *NVD*, followed by 6.6 on *Snyk*. In other words, *NVD* has an average CVSSv3 rating that is 0.8 higher than on *Snyk* for the matching security reports.

In order to present a more detailed view on the distribution of the scores, we created three plots. Figure 4.2, Figure 4.3, and Figure 4.4 follow the same structure. Their x-axes represent the indices of the entries, *i.e.*, each security report score was alphabetically sorted and then indexed from zero to the number of available entries. The y-axes indicate the CVSS scores. Although the security reports of the indices might not correlate, we can see the distribution of the assigned scores. When we focus on the differences between the CVSSv2 scores, we can clearly see that *RAPID7* very closely matches the distribution of *NVD*, as shown in Figure 4.2. The contrary is true for *Snyk* and its CVSSv3 scores: it seems that numerous scores differ from *NVD*, as seen in Figure 4.3.

When we inspect the 120 014 entries within *NVD*, *NVD* is the only database that contains CVSS scores for both versions 2.0 and 3.0; we found an increase for the averaged scores of about 1.3 points (22%), *i.e.*, from a CVSSv2 average of 6.0 to a CVSSv3 average of 7.3. Surprisingly, 3 447 (3%) entries kept their values, while 116 567 entries had their value changed upon migration, mainly increased. When we investigate the distribution of the scores in Figure 4.4, we can observe that the diversity of the CVSSv3 scores has increased once again (as from CVSSv1 to CVSSv2 [24]). However, it seems that at the end of the spectrum the distribution became worse: 23 661 entries received in CVSSv3 a score of 9.8 compared to 6 759 entries that received in

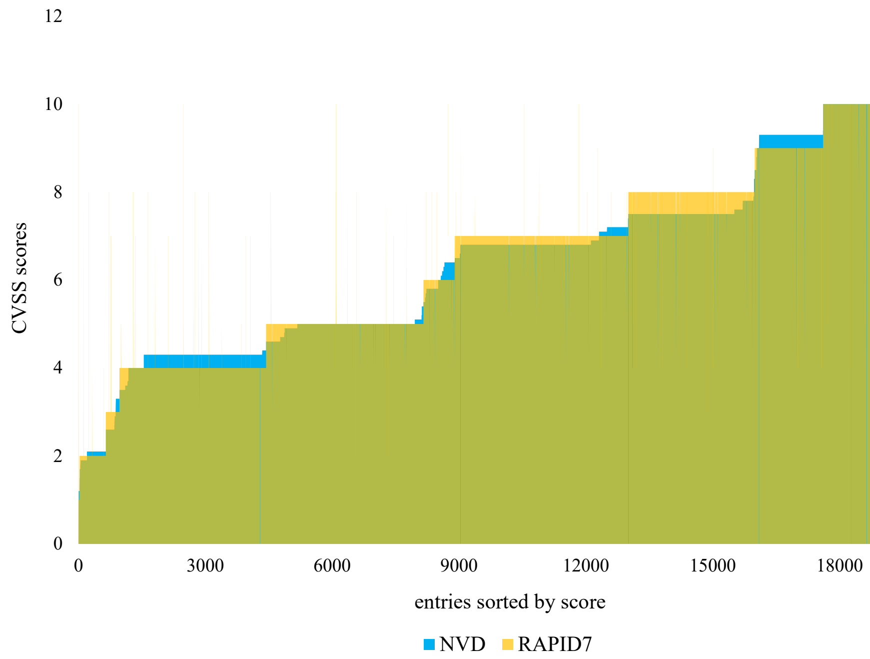

Figure 4.2: CVSSv2 scoring differences between two different operators

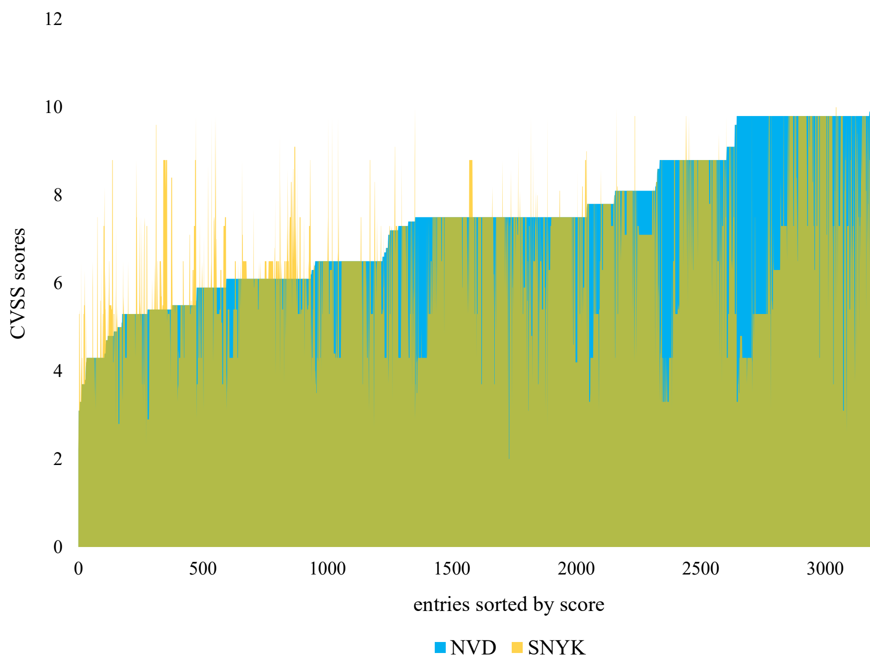

Figure 4.3: CVSSv3 scoring differences between two different operators

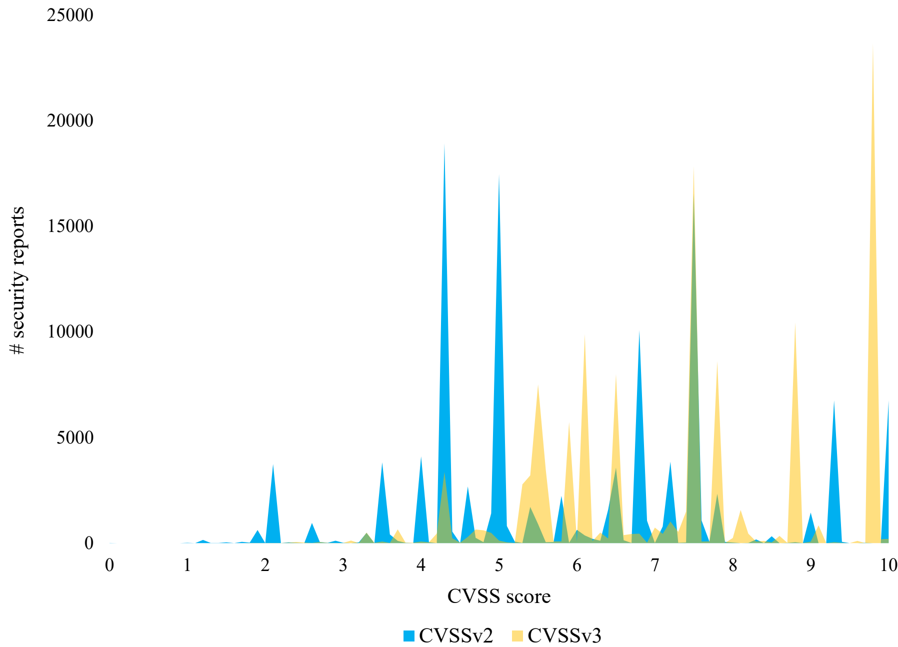

Figure 4.4: Scoring differences between CVSSv2 and CVSSv3

CVSSv2 a ten (and none with 9.8). In addition, no entry received in CVSSv3 a score of 0.0, compared to three with CVSSv2.

Although the CVSS scheme intends to establish comparability between reports and databases, this has not yet been achieved. Without in-depth manual investigations it is impossible for users to compare scores. We see a need of transparency for the algorithms and review guidelines that lead to the provided scores. Future research should decide whether the vulnerabilities at the end of the spectrum are rated accurately.

4.5 Trends

In this section, we try to answer the question *Can we predict trends based on the collected data?* We discuss three trends found during the analysis of collected data, based on the comparison of term frequencies.

4.5.1 Shared Code

Software built upon shared code is common for large well-designed modular projects, where essential features are shared across applications. Such projects that share common code, consequently, can also share their vulnerabilities. Although this might not be the case for every vulnerability, *e.g.*, some vulnerabilities may require an interplay

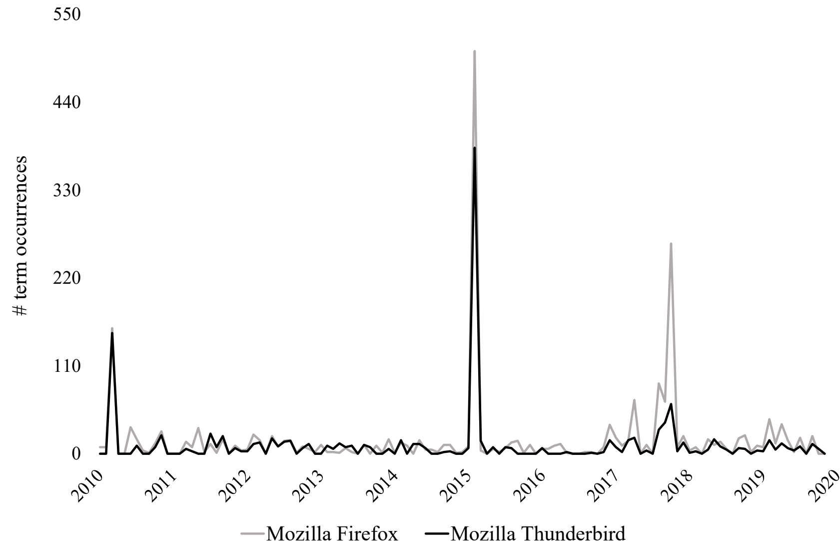

Figure 4.5: RAPID7 security report publications about shared code

with other parts of the system, we argue that the risk is still considerable due to the code reuse. A typical example is that of the web browser *Mozilla Firefox* and the email client *Mozilla Thunderbird*. They both share much code which massively simplified their development. For example, they both use an HTML renderer, *i.e.*, Thunderbird to display HTML emails and Firefox for websites. Moreover, both of them support user installable extensions and UI customization.

In Figure 4.5 we show their corresponding security report frequency. The x-axis denotes the date the product name appeared in security reports, whereas the y-axis illustrates the number of occurrences. We can see that many of the smaller peaks correlate very closely to each other. We manually verified that security reports covered related issues where the peaks coincide; in many cases security reports even mentioned both applications in the descriptions. The two largest peaks are caused by *RAPID7*'s binge-processing of entries for a period of time. We also investigated the submission rate for *Mozilla SeaMonkey* that also shares code with Firefox. The reporting frequencies between Firefox and SeaMonkey showed a similar behavior until 2015, but unfortunately SeaMonkey's popularity started to decline after that year and consequently the number of submitted security reports. Moreover, *Apple Safari* and *WebKit* also share substantial similarities, but surprisingly *Chromium* and *WebKit* to a much lesser extent.

4.5.2 Shared Libraries

Shared libraries naturally induce risks in clients using them. A traditional shared library use can be found in *Android*. An adapted version of *FFmpeg* provides the decoding

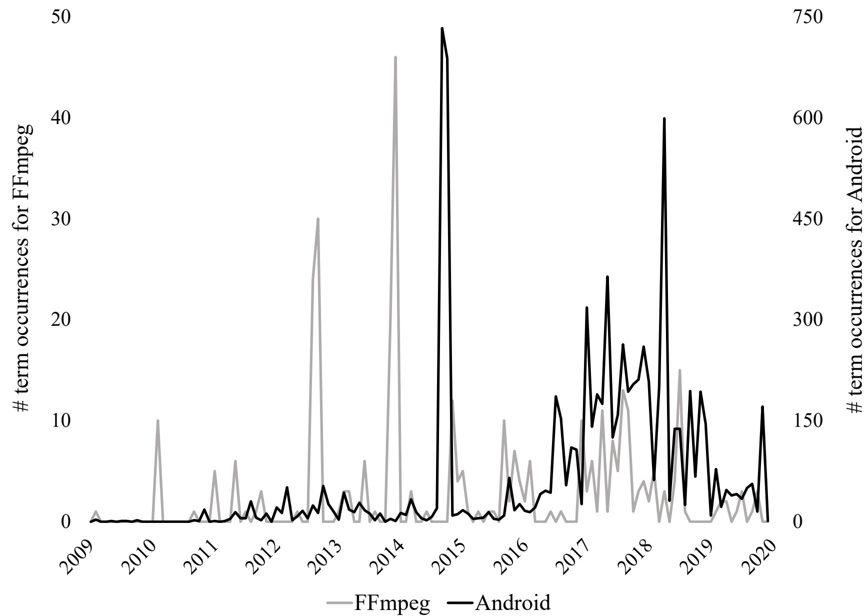

Figure 4.6: NVD security report publications about a shared library

routines for videos, music, and images to the *Android* mobile operating system. We selected these two products, because both terms appeared frequently in our list, which was not the case for other libraries.

The comparison is depicted in Figure 4.6. The x-axis denotes the date the product name appeared in security reports, whereas the y-axes illustrate the number of occurrences. To better visualize the similarity, two separate vertical axes were defined: the left axis showing the term frequency of *FFmpeg*, and the right axis for *Android*. We can see time-displaced similarities, especially for the large peak in DEC-2013 for *FFmpeg* that appears to correlate with the peak in SEP-2014 for *Android*. When we further investigate that peak, we find that *FFmpeg*'s peak is comprised of decoder vulnerabilities caused by unchecked passed values. On the other hand, *Android*'s peak is comprised of vulnerabilities based on vulnerable SSL implementations. As a result, we can say that these peaks are not related. Again, considering vulnerabilities before 2013 and after 2016, there is no correlation between peaks found in *FFmpeg* and *Android*. Besides, we conjecture that the low prevalence of the term *Android* before 2011 may be caused by the low distribution of *Android*-powered devices at that time, and the resulting lack of interest for security research. There exist more cases of applications and used libraries yielding plots unrelated to each other. One such example is the *Atom* text editor which uses the *Node.js* library through the *Electron* framework, but does not share major trends. We found that vulnerabilities in libraries usually are not reported

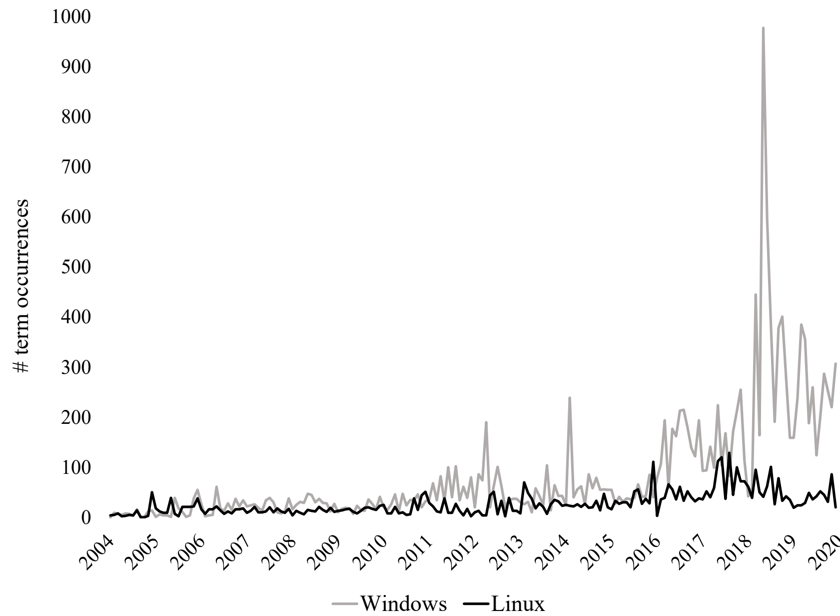

Figure 4.7: NVD security report publications about related software

separately for affected applications as a measure to reduce duplication. Instead, such applications are listed under "affected software." This explains why we did not find any similarities between the reports of applications and used libraries. Nevertheless, for major or complex vulnerabilities additional reports may still be submitted.

4.5.3 Related Software

We were interested to see if related software suffers from similar vulnerabilities, due to corresponding code logic. In order to compare related software, we chose two related products both with numerous term occurrences in our dataset, *i.e.*, the operating systems *Linux* and *Microsoft Windows*.

We show in Figure 4.7 data from *NVD* and in Figure 4.8 data from *Exploit-DB*. In both plots the x-axis denotes the date the product name appeared in security reports, whereas the y-axis illustrates the number of occurrences. As we can see in the *NVD* plot, the term frequencies for *Linux* and *Windows* are dissimilar, although both have increased activity in 2016 and beyond. Particularly interesting is the large peak for *Windows* in 2018. At the root of the cause are issues regarding the *Windows kernel* including the *Win32k* subsystem, and *Windows Device Guard* which prevents malicious code from running. Many of these issues lead in *Windows 7* and higher to information disclosure and elevation-of-privilege vulnerabilities. Surprisingly, the *Exploit-DB* plot does not reflect the increase in later years. We found that *Exploit-DB* cherry picks

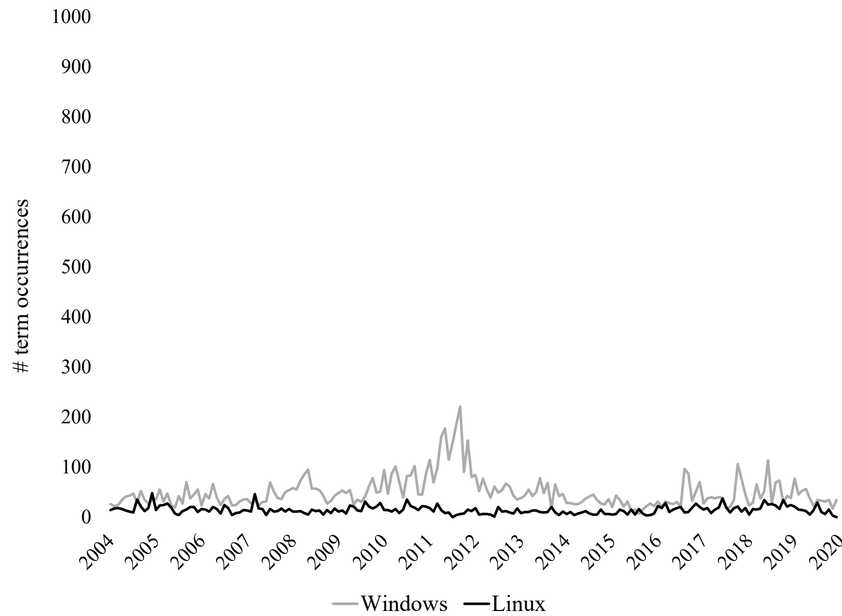

Figure 4.8: EDB security report publications about related software

security reports that have an accompanying proof-of-concept and are different enough to warrant their own entry.

When we only focus on one product, we see repetitive activity cycles. It seems that those cycles correspond to the product's release cycle. In fact, the first peak for *Windows* (APR-2012) is twelve months after *Windows Home Server 2011* was released, the second peak (FEB-2014) is four months after the release of *Windows 8.1* and *Windows Server 2012 R2*, and the third large peak (MAY-2018) is less than one month behind the release of *Windows 10 1803*. For *Linux* we do not see that much activity, and it only partially correlates with its release cycles, *e.g.*, DEC-2015 with release 4.4 LTS. Since Linux has a rolling release cycle, this might impact the results. Finally, if we compare *Apple* to *Microsoft* and *Linux*, we can spot some similarities, but this might be coincidental.

We can say that similarities from shared code bases can be detected, but generally not from used libraries. Similarities between related software are rather superficial. In addition, we can determine the used release type, *e.g.*, rolling release, or less frequent major releases.

5 Outlook

Reports are on the rise and database providers are not always able to keep up with the rate of new reports being submitted. This leads to occasional peaks that not always accurately reflect dates of submissions. Moreover, the security report data is nowadays used for many different services like security consulting, manual and automated security audits, or IDE feedback during development. Such different use cases have not yet received much support from the databases. For instance, if a security investigator requires some vulnerability information, many different sources must first be manually traversed. This is clearly inefficient and not desired.

In the future we would like to see databases that more transparently build upon existing major databases like *CVE* or *NVD*, instead of creating replicas that are hard to maintain for a rather small team. Such an improved database should also consistently support features like CVE-ID, vulnerability information, consistent dates, and references to other vulnerabilities or sources. Existing CVSS scores should be used uniformly, and if certain scores are not accepted, they should be augmented with comments instead of being silently replaced. Wherever possible, a proof-of-concept (text, script, or code) and remediation strategies for the vulnerability should be provided. This task, for example, could be performed by security researchers crawling the web for additional information that has not yet been provided. It should be straightforward to see which software is vulnerable and the mitigation strategies should be clear and not generic, *i.e.*, "update software" does not provide the necessary details to successfully

mitigate the risk. APIs of such vulnerability databases should be integrated into update services of software, such that an update client automatically detects a security report and disables the affected module until an update becomes available.

Ultimately, we believe that future databases require support for remotely stored data that can be traced back to the originator, and that they need broader feature support, *e.g.*, a common property “exploit code” or an interface accessible by update services on client devices.

6
Threats to Validity

In this chapter we discuss the threats to validity that might affected our results.

6.1 Completeness

A major threat to validity is the sampling bias of this study as we limited our focus to five major database providers. We strived to collect all entries without any preemptive filtering, while including all the information we could get. However, we cannot guarantee that all relevant information was extracted from websites, nor that all relevant data was published in the first place. We ignored submissions released or updated after our crawling process.

6.2 Correctness

We did not validate the accuracy of entries maintained by different database operators. Data used in security reports might be inaccurate due to rogue out-of-order disclosures, synchronization inaccuracies between providers, or human error. Furthermore, our regular expression-based HTML scraper might suffer from flaws, especially in regards to foreign characters and non-alphanumeric symbols. We excluded all invalid, cleared, or unpublished entries.

Our cross-referencing could suffer from mismatches: each database has different properties that require manual workarounds to successfully map the corresponding key-value data pairs. We found undocumented data in databases that we ignored. The terms used for the analysis of trends were hand-picked by the authors and their occurrence may be unrelated to the affected product, *e.g.*, "Windows" is also used in descriptions from software designed for that operating system. However, only the most prevalent terms have been considered for the study, and we sampled a few entries for each used term to ensure they match the corresponding product.

Due to our use of an adapted English-based stop-words filter list, it is possible that despite our efforts, we removed terms that could have been relevant for analysis. In order to mitigate this threat, we manually reviewed the list of the most frequent terms for the top 200 entries, before we applied the filter.

There is a threat to the validity of all our results and findings, through potential personal bias, as we are unable to completely withhold our opinions. Our expectancy of the results of the study may further cloud our process and findings.

7 Related Work

The question "How to properly design a vulnerability database?" has been investigated intensively by researchers. A new vulnerability database structure has been proposed by Yap *et al.*, with the aim to curate machine readable content that fosters automated actions, *e.g.*, notifications for security personnel, or patch generation [28]. Their solution collects the data from three different vulnerability databases, and with the help of machine-learning it extracts from the received data various properties, *e.g.*, hardware requirements or required privileges. Armold *et al.* propose a consolidation of information. They collect data from four different databases and build a new database that contains all information [1].

The accurate scoring of vulnerability threats requires a comprehensive assessment of its impact. Numerous vulnerability classification algorithms with different foci have been proposed and used in industry, *e.g.*, Krsul investigated 17 different measures in his work [10]. In 2005 with the metric CVSS, the National Infrastructure Advisory Council, *i.e.*, a United States government advisory council, proposed a global scoring framework to create uniform scores that are better understandable, more transparent, and comparable.[1] However, the initial CVSS version suffered from deficiencies, *e.g.*, many final scores do not reasonably correlate with the threat potential, and minor score nuances are lost in the process. This leads to many scores that are identical unlike their corresponding risk [13]. Scarfone *et al.* reevaluated some of those deficiencies and

[1] `https://www.first.org/cvss/v1/cvss-dhs-12-02-04.pdf`

found that the revised version leads to a higher average score and a greater score diversity [24]. According to them, the more granular properties added to the new release only have a relatively small effect on score diversity, thus they might be not worth the effort. Most importantly, they recommend that any future changes to CVSS should be validated with experimental data rather than theoretical data, because the results may differ significantly. In the meantime, many researchers published other approaches that improve particular aspects of CVSSv2, *e.g.*, using different weights [25], different algorithms [12], temporal features such as exploit tool availability [9], or automated machine-learning to compute the final scores [6]. We did not find any comprehensive work on CVSSv3 or CVSSv3.1.

For industry, data from vulnerability databases have been in focus for risk assessment techniques throughout academia. For example, Houmb *et al.* build a predictive risk model for software based on publicly available CVSS scores [5]. They exemplify the model using a safety- and mission-critical system for drilling operational support which allows the operators to take immediate action in case of suddenly increased predicted risks. Traditional software companies can leverage such data as well: Joh *et al.* develop methods to assist managers with software selection and patch application decisions in a quantitative manner [7].

More recently, researchers started to use the reports for benign or malicious code generation. Benign code generation can be seen in the automatic patching of reported vulnerabilities. Liu *et al.*, for instance, developed an approach to automatically fix bugs based on the free-form text description of bug reports [11]. Their implementation is able to create patches for the Linux kernel. On the other hand, generative methods can be misused for attack preparation or execution, *e.g.*, by automatically creating exploits based on reported vulnerabilities. Brumley *et al.* were successful in generating patch-based exploits for commercial software [2]. Based on the received patches from *Windows Update*, that is *Microsoft*'s software update service, they performed a code diff analysis on the patched files and successfully reconstructed exploits from the found differences.

8
Conclusion

The vulnerability databases' affiliations and contributions are non-trivial and have not yet been studied in depth. We collected publicly available data from the websites of five major vulnerability database operators, before we normalized and correlated the individual entries to track them within different vulnerability databases. We focus on the affiliations between the different operators, their feature support and consistency, and we share interesting discoveries we made during our studies. 370,298 security reports were extracted, 89% of them were accessible at more than one provider.

Surprisingly, many reports were inconsistent with respect to scores and descriptions. We found that three of five operators support scores, and that scores can on average diverge up to 0.8 points depending on the operator, and up to 1.3 points depending on the used CVSS version. In addition, after the CVSS migration from version 2.0 to version 3.0, 3% of the scores remained unchanged in *NVD*. Contrary to our belief, content across databases differs not only by references but also details for identical CVE-IDs. For instance, some operators provide a description, while others, instead, provide the exploit code, or other relevant information. This makes the issue tracking over database boundaries unnecessarily cumbersome for security personnel.

When inspecting term frequencies over time, it is possible to determine common code bases, but no library usage through security reports. Moreover, release cycles might be recoverable, *e.g.*, whether a product is developed using a continuous release strategy, or following a major release schedule. Based on our experiences, we list

several features that a "perfect" database would support. This might spark further work on how to improve existing vulnerability databases.

Bibliography

[1] A. Armold, B. Hyla, and N. Rowe. Automatically building an information-security vulnerability database. In *2006 IEEE Information Assurance Workshop*, pages 376–377. IEEE.

[2] D. Brumley, P. Poosankam, D. Song, and J. Zheng. Automatic patch-based exploit generation is possible: Techniques and implications. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 143–157. IEEE.

[3] Q. Chen, L. Bao, L. Li, X. Xia, and L. Cai. Categorizing and predicting invalid vulnerabilities on common vulnerabilities and exposures. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 345–354. IEEE.

[4] H. Holm and K. K. Afridi. An expert-based investigation of the common vulnerability scoring system. 53:18–30.

[5] S. H. Houmb, V. N. Franqueira, and E. A. Engum. Quantifying security risk level from CVSS estimates of frequency and impact. *Journal of Systems and Software*, 83(9):1622 – 1634, 2010. Software Dependability.

[6] G. Huang, Y. Li, Q. Wang, J. Ren, Y. Cheng, and X. Zhao. Automatic classification method for software vulnerability based on deep neural network. 7:28291–28298.

[7] H. Joh and Y. K. Malaiya. Defining and assessing quantitative security risk measures using vulnerability lifecycle and CVSS metrics. In *The 2011 international conference on security and management (SAM)*, pages 10–16, 2011.

[8] P. Johnson, R. Lagerstrom, M. Ekstedt, and U. Franke. Can the common vulnerability scoring system be trusted? A bayesian analysis. 15(6):1002–1015.

[9] M. Keramati. New vulnerability scoring system for dynamic security evaluation. In *2016 8th International Symposium on Telecommunications (IST)*, pages 746–751. IEEE.

[10] I. V. Krsul. *Software vulnerability analysis*. Purdue University West Lafayette, IN, 1998.

[11] C. Liu, J. Yang, L. Tan, and M. Hafiz. R2Fix: Automatically generating bug fixes from bug reports. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pages 282–291. IEEE, 2013.

[12] Q. Liu and Y. Zhang. VRSS: A new system for rating and scoring vulnerabilities. 34(3):264–273.

[13] P. Mell and K. Scarfone. Improving the common vulnerability scoring system. 2007.

[14] A. Nappa, R. Johnson, L. Bilge, J. Caballero, and T. Dumitras. The attack of the clones: A study of the impact of shared code on vulnerability patching. In *2015 IEEE Symposium on Security and Privacy*, pages 692–708. IEEE.

[15] ExploitSearch. Exploit search through advanced exploit search engine, 2020.

[16] First Inc. Information and resources on CVSS-SIG, 2020.

[17] Offensive Security. Exploits database, 2020.

[18] Rapid7. InsightVM; public vulnerability database among its resources, 2020.

[19] Snyk Ltd. Open source security solution; public vulnerability database among its resources, 2020.

[20] The MITRE Corporation. Common vulnerabilities and exposures database, 2020.

[21] The MITRE Corporation. Community-developed list of software weakness types, 2020.

[22] U.S. National Institute of Standards and Technology. U.S. national vulnerability database, 2020.

[23] Vulners Inc. search engine for vulnerability database contents from multiple sources, 2020.

[24] K. Scarfone and P. Mell. An analysis of CVSS version 2 vulnerability scoring. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 516–525. IEEE.

[25] G. Spanos, A. Sioziou, and L. Angelis. WIVSS: a new methodology for scoring information systems vulnerabilities. In *Proceedings of the 17th Panhellenic Conference on Informatics - PCI '13*, page 83. ACM Press.

[26] M. ur Rahman, V. Deep, and S. Multhalli. Centralized vulnerability database for organization specific automated vulnerabilities discovery and supervision. In *2016 International Conference on Research Advances in Integrated Navigation Systems (RAINS)*, pages 1–5. IEEE.

[27] D. Waltermire, S. Quinn, H. Booth, K. Scarfone, and D. Prisaca. The technical specification for the security content automation protocol (SCAP) version 1.3.

[28] R. H. C. Yap, L. Zhong, and Q. International. A machine-oriented integrated vulnerability database for automated vulnerability detection and processing. page 12.

Anleitung zum wissenschaftlichen Arbeiten

The Anleitung consists of the conference paper "An Investigation into Vulnerability Databases".[1]

P. Gadient, B. Schweigler, M. Ghafari, and O. Nierstrasz. An investigation into vulnerability databases.

Planned for submission to ICSME *The 36th IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020.

[1] http://scg.unibe.ch/download/supplements/Investigation-Vulnerability-Databases-(working-draft).pdf