Evaluating the Data Inconsistency of Open-Source Vulnerability Repositories

Yuning Jiang
University of Skövde
Skövde, Sweden
yuning.jiang@his.se

Manfred A. Jeusfeld
University of Skövde
Skövde, Sweden
manfred.jeusfeld@his.se

Jianguo Ding
University of Skövde
Skövde, Sweden
jianguo.ding@his.se

ABSTRACT

Modern security practices promote quantitative methods to provide prioritisation insights and support predictive analysis, which is supported by open-source cybersecurity databases such as the Common Vulnerabilities and Exposures (CVE), the National Vulnerability Database (NVD), CERT, and vendor websites. These public repositories provide a way to standardise and share up-to-date vulnerability information, with the purpose to enhance cybersecurity awareness. However, data quality issues of these vulnerability repositories may lead to incorrect prioritisation and misemployment of resources. In this paper, we aim to empirically analyse the data quality impact of vulnerability repositories for actual information technology (IT) and operating technology (OT) systems, especially on data inconsistency. Our case study shows that data inconsistency may misdirect investment of cybersecurity resources. Instead, correlated vulnerability repositories and trustworthiness data verification bring substantial benefits for vulnerability management.

ACM Reference Format:
Yuning Jiang, Manfred A. Jeusfeld, and Jianguo Ding. 2021. Evaluating the Data Inconsistency of Open-Source Vulnerability Repositories. In *The 16th International Conference on Availability, Reliability and Security (ARES 2021), August 17–20, 2021, Vienna, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3465481.3470093

1 INTRODUCTION AND BACKGROUND

The cumulative reported vulnerabilities and the potential to exploit them pose severe challenges to the cybersecurity research community. For example, the recently disclosed security holes in Microsoft Exchange may lead to at least 26 000 exchange servers in Germany being directly accessible from the Internet. This vulnerability is disclosed in the Common Vulnerability Exposure (CVE)[1] repository with an ID as CVE-2021-26855. CVE database is one of the most influential forces in sharing and standardising vulnerability information under the cybersecurity community efforts. CVE repository discloses more than 150 000 reported and exploited vulnerabilities from 1999 until March 2021. It provides standardised but unstructured textual descriptions of vulnerabilities, which are fed into the National Vulnerability Database (NVD)[2] for further vulnerability assessment like vulnerable system configurations, weakness categories and common vulnerability severity scores (CVSS)[3].

1.1 Background

Modern security practices promote quantitative methods to provide prioritisation insights and support predictive analysis supported by cybersecurity databases. Research efforts are seen in vulnerability or threat feature extraction (like malware signatures) [19], and cyber intelligence collection [10][8] from public vulnerability repositories and technical blogs. Besides, some commercialised vulnerability management and assessment products, such as *InsightVM* [4] from Rapid7 Research, *Nessus* [5] and *Tenable.io* from Tenable, deploy system scans that are matched with CVE repository to monitor vulnerabilities and exposures of infrastructures. Both of these research and commercial solutions heavily rely on integrating multiple open-source cybersecurity information, which makes it critical to resolve conflicts and establish the correctness of such information.

Data quality of public cybersecurity data sources has raised concerns in the cybersecurity community [9]. Incomplete [3], outdated and incorrect vulnerability entries, may leave a risk window for potential zero-day attacks [13] [14][13]. Public vulnerability databases heavily rely on manual reporting like the CVE Numbering Authorities (CNA) [6] led process and further analysis, which leaves room for potential errors [15]. For instance, around 25% CVE reports that involve Google Chrome as affected products have incorrect Chrome version strings [14]. Besides, NVD, VulDB[7], and other similar repositories that extract data feeds from CVE may provide contradicting analysis for reported vulnerabilities [1].

1.2 Related Works

Data inconsistency researches are primarily focused on the identification of inconsistencies [9] and truth discovery of multiple conflicting information that is provided in different websites [17].

Nappa et al. spent substantial efforts to identify and report discrepancies in NVD vulnerability entries. They crawled online security advisories in *Mozilla* website and compared extracted vulnerable version ranges published in NVD. They summarised three

[1] https://cve.mitre.org/index.html

[2] https://nvd.nist.gov/vuln
[3] https://www.first.org/cvss/
[4] https://www.rapid7.com/products/insightvm/
[5] https://www.tenable.com/products/nessus
[6] https://cve.mitre.org/cve/cna.html
[7] https://vuldb.com/

ARES 2021, August 17–20, 2021, Vienna, Austria

ACM ISBN 978-1-4503-9051-4/21/08…$15.00
https://doi.org/10.1145/3465481.3470093

main inaccurate issues of NVD entries, namely missing vulnerable versions, extraneous vulnerable versions, and missing vulnerable program file versions. The third issue is particularly true for Microsoft products such as *msword.exe*, according to the authors [13]. Dong et al. conducted quantitative analysis on vulnerable software version inconsistencies between CVE and NVD. Their experimental data include vulnerabilities reported from January 1999 to March 2018, which indicates that only 59.82% of the CVE summaries strictly match with the standardised NVD entries [4]. They also compared the matching rate of vulnerable software versions between NVD against CVE and five other public cybersecurity data sources, namely OpenWall, Security Focus, Security Focus Forum, Security Tracker, and ExploitDB. The result shows that the matching rate between NVD and ExploitDB is the highest, followed by the matching rate between NVD and CVE.

Jo et al. developed a text-mining technique based tool, GapFinder, to identify semantic inconsistencies and technical inconsistencies of open-source malware threat reports [9]. By combining malware graph constructor with named entity recognition (NER) and relation extraction (RE) language processing techniques, their tool also addresses various language-specific issue and malware domain-specific issues. Anwar et al. perform a systematic evaluating of the consistency and completeness of NVD data feeds, based on which they further improve the quality of the data [1]. Farhang et al. [5] have investigated the degree to which Android smartphone vendors have included vulnerabilities in their security bulletins and found significant differences in the timeliness of reporting and the consistency of the reports.

However, to be best of our knowledge, there has not been a study investigating the implications of data inconsistencies of open-source cybersecurity repositories from the perspective of computing-system users. In this paper, we conduct a case study of a real-world IT/OT system to examine the impact of data inconsistencies in vulnerability repositories on cybersecurity prioritisation decision-making. We also investigate the possible factors contributing to data inconsistencies, which provides insights for the cybersecurity community on the curation of online security data sources.

1.3 Problem Statement

Data quality dimensions are used to guide the measurement of data validation. Commonly used data quality dimensions include and are not limited to accuracy, completeness, uniqueness, timeliness, currency, and consistency [7, 11]. In this paper, we focus on the issue of data inconsistency. We adopt the definition from David Loshin that data inconsistency refers to data values in one data set not being consistent with values in another data set [11]. The research questions we explore in this paper are: do multiple open public cybersecurity data sources provide consistent information regarded to vulnerability analysis? Furthermore, how to evaluate the impact of data inconsistencies of these cybersecurity repositories?

In this paper, we aim to investigate the impact of data inconsistencies in open-source vulnerability repositories for real-world IT/OT infrastructures. In this case study, we match the system scan against a set of public vulnerability databases like CVE, NVD and vendor security repositories. By doing so, we extract component-based vulnerability lists for our investigated system, based on which we identify inconsistencies of retrieved vulnerability information. As a result, the gap between theoretical vulnerabilities in open-source vulnerability repositories and the actual systems is measured. Further, the implication of our findings is discussed with domain experts to solidify the results. More specifically, we interviewed three IT technicians from one organisation in Sweden and two cybersecurity experts from other organisations. Deep analysis of conflicting vulnerability reports also indicates important insights and guidance for the cybersecurity community on the usage of online public vulnerability data sources.

The rest of this paper is organised as follows. In Section 2, we define the concept of data inconsistency, followed by an introduction of the major types of data inconsistency issues in vulnerability repositories. Section 3 discusses the employed data sources and the targeted vulnerability attributes in this paper, along with detailed guidance of our inconsistency identification approach. In Section 4, we present a case study whereby we perform a vulnerability assessment of actual IT/OT systems to validate the impact of data inconsistencies. We show the results of this case study through extracted statistic patterns and summarise our interview evaluation, followed by some discussion on limitation and key insights. We give some concluding remarks and future research directions in Section 5.

2 DATA INCONSISTENCY DEFINITION

Given a list of n vulnerability data sources $[V_1, \cdots, V_i, \cdots, V_n]$ $(0 < i \leq n)$, when we query data source V_i, we generate a new set of k vulnerabilities $V_i' \subseteq V_i$. For each vulnerability $v_{i,j} \in V_i'$ $(0 \leq j \leq k)$, it has a set of attributes $a_p \in A$ $(0 < p \leq m)$. Therefore, each vulnerability instance $v_{i,j}$ has a vector $v_{i,j}$=$[v_{i,j}^{a_1},\ldots,v_{i,j}^{a_p},\ldots,v_{i,j}^{a_m}]$. According to David Loshin, consistency analysis include 5 contexts, namely record-level, cross-record, temporal, application/business-level, and reasonableness-level consistency [11]. In this paper, we mainly consider record-level and cross-record data inconsistencies.

2.1 Record-Level Inconsistency

Record-level inconsistency exists between vulnerability attributes $v_{i,j}^{a_{p1}}$ and $v_{i,j}^{a_{p2}}$ $(0 < p1 \leq m, 0 < p2 \leq m, p1 \neq p2)$. Record-level inconsistency is seen where multiple names are provided to represent the same entity, such as vendor names, vendor-product names, and vulnerable product versions [1]. Vendor names are not identical in CPE metadata due to various reasons such as misspelled names (e.g. *Schneider Electric* has been spelled as *'schneider-electic'* and *'chneider-electric'*), and using abbreviated names (e.g. *General Electric Company* has been expressed as *'ge'* and *'general-electric'*). The causes for inconsistent vendor product names are similar to the ones for vendor names, but is also related to the fact that different stakeholders may provide different names for the same product. Record-level inconsistency appears when different attributes of the same vulnerability provide contradictory information.

2.2 Cross-Record Inconsistency

Cross-record inconsistency exists between vulnerabilities $v_{i1,j}^{a_p}$ and $v_{i2,j}^{a_p}$ whereby $v_{i1,j} \in V_{i1}, v_{i2,j} \in V_{i2}$ $(0 < i1 \leq n, 0 < i2 \leq n)$. This inconsistency indicates scenarios where the same attributes

such as vulnerability severity scores, publication dates and vulnerable products are conflicting, or where multiple attributes indicate contradicting vulnerability characteristics.

Different data-repository sources may provide conflicting severity scores. For example, the vulnerability CVE-2015-6461 is assigned CVSS V2 base score of 5.5 by NVD and 3.2 by ICS-CERT. The score inconsistency arises due to contradictory conclusions on the access vector of this vulnerability, for which NVD assigns it as network-based and ICS-CERT assigns it as local-based. Under CVE-2017-6023, this vulnerability is assigned CVSS V3 base score of 9.8 with vector *AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H* by NVD and 7.3 with vector *AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:L* by ICS-CERT, separately.

Publication dates indicate the time when vulnerabilities become public and provide essential tracks for system protection prioritisation. Nevertheless, inconsistencies of such publication dates occur commonly due to disagreement between NVD publication dates and public disclosure dates on other vulnerability databases and cybersecurity blogs [1][9].

CPE[8] (referring to Common Platform Enumeration) vulnerable version ranges for each NVD entry can be inconsistent with identified vulnerable versions from third-party analysers like CERT Coordination Center [6] and vendors [4]. One example is CVE-2019-5527 that refers to a use-after-free vulnerability in the virtual sound device. According to the security advisory VMware, mitigation for this vulnerability is to upgrade the vulnerable component VMware ESXi to ESXi670-201904101-SG (this is a patch package name and is linked with build number 13006603). And hence, VMware ESXi versions later than ESXi670-201904101-SG are not affected by this vulnerability. However, those later versions like ESXi670-202004001 are listed as vulnerable versions in NVD.

3 DATA INCONSISTENCY ANALYSIS

In this section, we first briefly introduce the vulnerability data sources used in this paper. Then we present the process of data-inconsistency identification and validation in a step-by-step format.

3.1 Data Sources

We query related vulnerability instances of our investigated system from 3 main cybersecurity repositories, namely CVE (or V_{CVE}), NVD (or V_{NVD}, and include CPE entries), and Vendor websites (or V_{vendor}). MITRE Corporation publishes the CVE industry-standard to assign an identifier to each discovered vulnerability. It maintains a publicly accessible database of all identifiers. NVD is a vulnerability database built upon CVE. We obtain the vulnerability repositories by downloading the data feeds from CVE and NVD, and then parse and query them in the local computer terminal using Python. CPE based search capabilities are also accessed through NVD entry. Besides, we deploy vendor data as an additional source to add the perspective of manufacturers. We crawl vendor websites such as the Microsoft Security Response Centre (MSRC)[9] and the VMware Security Advisories[10]. We then fetch and parse extracted vulnerability data for affected products and version ranges. For instance, *Microsoft* security updates are accessible through an official API [11]. Additionally, there are open-source Github projects (e.g. VIA4CVE [12]) that query this API and fetch *Microsoft* bulletins and vulnerability data in JSON format. Note that NVD entries are structured, while CVE reports and crawled vendor texts are unstructured.

Attribute	CVE	NVD	Vendors
Description	Yes	Yes	Yes
Applicable Product	Yes	Yes	Yes
Version Range	Yes	Yes	Yes
CVSS Score	N/A	Yes	Some
Impact Element	N/A	Yes	Some
Weakness Type	Some	Yes	Some

Table 1: Deployed Vulnerability Data Sources

A vulnerability is defined by the Society for Risk Analysis (SRA) [2] as "*the degree a system is affected by a risk source or agent; or the degree a system can withstand specific loads; or the risk conditional on the occurrence of a risk source/agent; or the uncertainty about and severity of the consequences, given the occurrence of a risk source*". Based on this definition, combining multiple attributes, such as the *criticality* of vulnerable components, the *severity* of emerging *vulnerability* instances, as well as the *likelihood* of exploiting attacks, contributes to the computation of the *vulnerability index*. The *criticality* property is weighed with attributes like the applicable product, version range, and functionality importance of the vulnerable component. The *severity* property is measured with attributes like CVSS score and impact elements (or sub CVSS metrics like confidentiality impact). The *likelihood* property is estimated with attributes like attack vector, attack complexity, exploit development status, and remediation development status.

In this paper, we aim to compare inconsistencies between CVE, NVD, and vendor repositories. Therefore, we focus on six attribute sets for vulnerability analysis that are available in a minimum of two data sources, as depicted in Table 1. Description typically covers the applicable product, version range, the vulnerable function, and sometimes the weakness type. Among them, the applicable product and version range can be identified manually or extracted using an open-source NER (referring to Named Entity Recognition) model [16]. CVE, NVD, and vendor websites provide accesses to such vulnerability descriptions. The applicable product and version range are also easily identified and extracted from the CPE metadata. Note that the CPE metadata also indicates the applicable vulnerable system configurations. For example, vulnerability instance *CVE-2020-0964* describes a remote code execution vulnerability in Windows Graphics Device Interface, which applies to a list of servers like Windows Server 2016 and Windows Server 2019. Another vulnerability instance *CVE-2020-0966* indicates a remote code execution vulnerability in the VBScript, which is only applicable to Window Server 2016 and 2019 when Internet Explorer 11 is running on the server. Therefore, accurate and complete system configuration information is needed to extract relevant vulnerability instances.

[8] https://cpe.mitre.org/
[9] https://msrc.microsoft.com/update-guide/vulnerability
[10] https://www.vmware.com/security/advisories.html
[11] https://github.com/microsoft/MSRC-Microsoft-Security-Updates-API
[12] https://github.com/cve-search/VIA4CVE

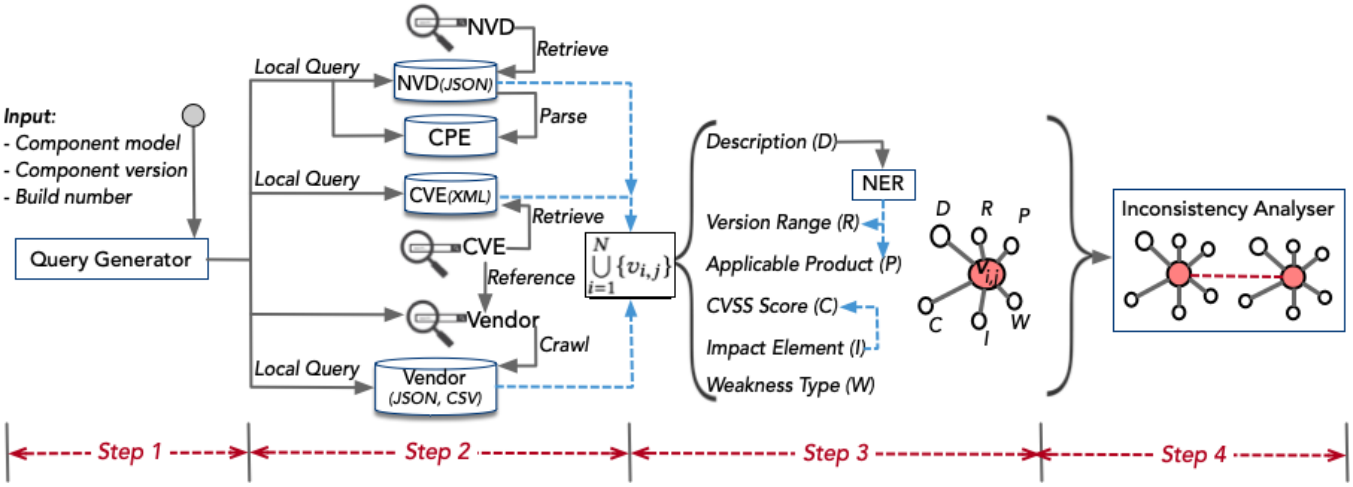

Figure 1: Overview of the Data Inconsistency Analysis Process

3.2 Data Inconsistency Analysis Process

The process of data inconsistency analysis is composed of four main steps, as illustrated in Figure 1.

Firstly, we collects system configuration information and component details. We then write an import filter to translate component information into query tags. Data items like models, versions and the latest patching update (e.g. cumulative security update KB, or knowledge base, package numbers) are necessary for vulnerability retrieval from online public vulnerability data sources. Due to software upgrade versioning, version numbers directly extracted from computer specifications are usually release versions. Yet, release versions may not reflect the installed update-package numbers of the component or the internal-version numbers from the vendors. One example is the *VMware vCenter Server* with internal version numbers from the earliest version *virtualCenter 2.5.0 GA* to the latest version *vCenter Server 7.0 update 1d (7.0.1.00300)*, till March 8th 2021. The release version *vCenter Server 6.0* has 18 different internal version numbers, each of which is related to various release updates. For instance, given component model "*VMware vCenter server*", version number "*6.0*", and build number "*9109103*", we check the internal version number as "*u3g*"and generate query tags like *type: software*, *vendor: VMware*, *product: vcenter_server*, *version: 6.0:u3g* in a format similar to the CPE match-list metadata.

Secondly, the translated tags from the previous step were used in Python-based queries in the PC terminal to extract component-based vulnerability instances. These tags are matched against the CPE metadata to retrieve corresponding vulnerability instances from NVD. To do so, we first convert the downloaded disk-based JSON files from NVD into a Python Dictionary data structure that organises the data as key-pair values. Some values are scalars (e.g. CVSS entries), while others can be either a list of other dictionaries (e.g. CPE entries). We then match the query tags against CPE entries of each vulnerability instance and return the instances that contain these tags. We parse the description entry of CVE XML files to extract matching vulnerability instances. This manner is further composed of multiple traditional text processing sub-steps such as tokenisation and removing stop-words [18], as well as word matching. Similar data preprocessing process is applied to crawled vendor reports in CSV or JSON format. To fetch vulnerability reports from vendor websites, we employ two approaches. We locate the vendor-related URL references of disclosed vulnerabilities in CVE through reference maps, based on which we fetch the contents of the referred websites. The other method is that, given a vendor product, we directly connect to and request the vendor website to fetch vulnerability-related data. We then take the union of all retrieved vulnerability reports $\bigcup_{i=1}^{N}\{v_{i,j}\}$, or $V_{CVE} \bigcup V_{NVD} \bigcup V_{vendor}$.

Thirdly, we extract attributes listed in Table 1 for each vulnerability instance. By doing so, one vulnerability instance has a vector of six attributes $v_{i,j}=[v_{i,j}^{a_1},\ldots,v_{i,j}^{a_p},\ldots,v_{i,j}^{a_6}]$ $(0 < p \leq 6)$, where each attribute has a list of up to 3 values. We employ two approaches to obtain these values. We directly parse the NVD JSON files for the targeted attributes like affected products and version ranges. We then build a NER module based on an open-source project [13] and pre-trained token embeddings from another open-source project [14]. The latter project is trained on data from *CVEDetails*[15] website. In doing so, we annotate applicable product and version range in vulnerability descriptions extracted from CVE reports as well as crawled vendor text files. For example, vulnerability report for the instance *CVE-2017-1442* is assigned with PoS (referring to part-of-speech) tags where AP refers to application product, VR refers to version range, and 0 refers to other information: "*(IBM, AP) (Emptoris, AP) (Services, AP) (Procurement, AP) (10.0.0.5, VR) (is, 0) (vulnerable, 0) (to, 0) (cross-site, 0) (request, 0) (forgery, 0) (which, 0) (could, 0) (allow, 0) (an, 0) (attacker, 0) (to, 0) (execute, 0) (malicious, 0) (and, 0) (unauthorized, 0) (actions, 0) (transmitted, 0) (from, 0) (a, 0) (user, 0) (that, 0) (the, 0) (website, 0) (trusts, 0).*"By doing so, we create two new data sources for vulnerable product and version ranges from unstructured CVE reports and vendor reports, besides the ones

[13] https://github.com/crownpku/Information-Extraction-Chinese
[14] https://github.com/pinkymm/inconsistency_detection
[15] https://www.cvedetails.com/

directly parsed from NVD JSON files. We apply similar NER-based methods to extract weakness type in vulnerability reports.

Fourthly, we compare the attribute sets of each vulnerability instance set to diagnose inconsistencies. We extract the statistic patterns based on vulnerability instances retrieved from different data sources, to assess the degree of data inconsistencies. We then evaluate the impact of data inconsistencies of multiple online cybersecurity data sources in supporting vulnerability analysis. This validation is done by interviewing five cybersecurity experts. The interviews focus on two questions: 1) Does this case study deliver a significant number of reports for the investigated system and cover unknown weaknesses that fit the gap of vulnerability management? 2) How is the impact of the identified data inconsistencies in cybersecurity decision making, especially if the vulnerability analysis results can be misleading in priority patching? By answering these two questions, we quantify the influence of data inconsistencies of open-public cybersecurity repositories on vulnerability assessment and cybersecurity decision making.

4 CASE STUDY

We conducted a case study on vulnerability analysis of the IT and OT systems of a company in Sweden. Next, we briefly introduce the systems under investigation, followed by empirical analysis of the main causes and impact of data inconsistencies in the vulnerability-analysis result of our investigated systems. We focus on cross-record data inconsistency in this study.

4.1 Investigated System

The investigated IT system is composed of a DataCentre (containing both hardware and operating system layers), an application layer and a network layer, as depicted in Figure 2. In DataCentre, hardware components like physical servers integrate with operating system software and manage PC storage. DataCentre contains 14 components. For example, a middleware server bridges multiple partners' systems, while an USBHub server works as a USB network gate. Additionally, a hypervisor host deploys and serves virtual systems, which provides an abstraction layer for virtualisation. This virtualisation layer supports multiple hypervisors (or virtual machines), together with heterogeneous operating systems and applications that run in isolation.

The application layer includes the configuration of 3 system packages supported by hypervisors, namely the IoT (referring to the internet of things) system (containing 138 components), customer management system (containing 137 components), and control system (containing 27 components). Among these three systems, the control system enables monitoring and controlling of the physical system. The IoT system records physical process data and then sends collected information to the customer management system through the middleware server. The customer management system stores, analyses, and manages consuming data. In an IoT system, for example, 4 hypervisors are included to integrate guest operating system, access control (AC) server, encryption key-store (EKM) server, database (DB) server and application (APP) server. Note that servers within the same security zone are reachable to each other. Servers in different zones can communicate through specific access lists.

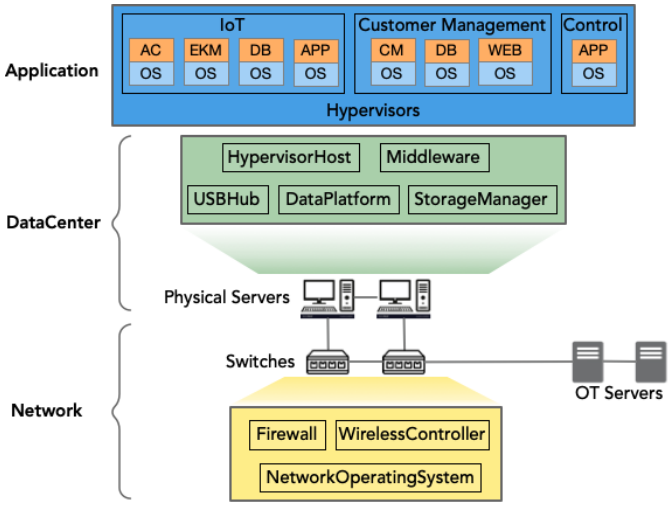

Figure 2: Structures of Investigated IT System

Physical servers in the DataCentre are connected to switches through fibres. The network layer represents the wireless connections of the IT network that contains 7 components. Some of the critical components are firewall, network operating system and wireless controller. These switches are further connected to the OT servers. The OT system is composed of 16 components that collect information and directly monitor physical processes.

4.2 Retrieved Vulnerability

The investigated system contains components provided by 18 different vendors, among which 10 vendors provide specific sections for security advisories. We found vulnerabilities for our system from all these 10 vendor websites. For the remaining 8 vendor data sources, we identified vulnerabilities related to 3 vendors from CVE and NVD. The distributions of vulnerability instances in different systems are listed in Columns 3-5 in Table 2. The network has comparatively more vulnerability instances per component, while DataCenter has the least vulnerability instances per component. We extract 562, 536 and 457 instances from CVE, NVD, and vendors separately. If we take the union of all the identified vulnerabilities, we have a total amount of 597 vulnerability instances. The amount of vulnerabilities changes to 435 when we take the intersection of the vulnerabilities, as listed in Columns 6 in Table 2. We observe that except for OT related vulnerabilities, the union of vulnerabilities retrieved from CVE and NVD cover the vulnerabilities extracted from the vendor websites. This observation only applies to this case study as the involved vendors are limited. On average, in the intersection set, the customer management system has the highest number of vulnerability instances per component. In contrast, the operating system has the least amount of vulnerability instances per component.

In this case study, we focus on cross-record data inconsistencies, especially between NVD and vendor advisories. CNA discloses vulnerabilities that are further analysed by NVD, vendor and other security analysers. CVE reports are deployed to provide ground truths in some studies that perform inconsistency detection [4]. In

Investigated System	Component Amount	CVE	NVD (CPE)	Vendor	Union	Intersection
IoT System	138	75	77	69	77	69
Customer Management System	137	309	319	311	319	309
Control System	27	50	53	28	53	28
OT	16	5	5	25	25	5
DataCentre	8	18	18	13	18	13
Network	7	105	75	11	105	11

Table 2: Retrieved Vulnerability Instances of The System

contrast, some other studies employ NVD entries as ground truths [13]. Nevertheless, the assumption of CVE or NVD as ground truth may not be valid. Instead, we try to explore the impact of relying on a singular data source for vulnerability analysis, especially between NVD and vendor repositories. The inconsistencies between different vulnerability report sources are reflected in the affected products and version ranges, weakness categorisation, exploit correlation, and vulnerability severity, which is discussed next.

4.3 Data Inconsistencies in Affected Products and Version Ranges

In this section, we discuss the impact and cause of discrepancies in terms of affected products and version ranges. Such discrepancies are reflected in the different amounts of identified vulnerabilities in Table 2. These discrepancies mainly result from inconsistent product names and different views on whether a specific component version is vulnerable. The network has the highest rate of inconsistent vulnerabilities in the IT system, followed by the IoT system and control system. Table 3 lists the vulnerability instances found in NVD and compares the results to the ones found in vendor entries. We also analysed if the component with the most vulnerabilities of each investigated system differs when involving different vulnerability data sources. We started with an investigation into the rate of data inconsistencies in each investigated system, as discussed next.

- The communication network system has the top data inconsistency rate among the six investigated subsystems, followed by the OT system. For example, in the OT system, the open platform communication (OPC) component has the highest amount of vulnerabilities when considering only NVD entries. When taking into consideration only the vendor entries, the protection and control component contributes the most vulnerabilities. The other four systems have relatively lower rates of data inconsistencies.
- In the IoT system, the database toolset component has the most vulnerabilities in the operating system, when considering only vendor entries. When accounting only NVD entries, the most vulnerable component in the operating system is the directory component. Similarly, the remaining four system servers, namely access control server, encryption keystore server, database server, as well as application server, have different rankings of the most vulnerable component after considering data inconsistencies.
- In the customer management system, the web server contributes an enormous amount of inconsistent vulnerabilities. The inconsistencies in the vulnerability sets of customer management system do not affect which component has the largest amount of vulnerabilities in each subsystem.
- In the control system, the application server has a higher rate of inconsistent vulnerabilities compared to the operating system. When accounting only entries from the vendor, the runtime library component has the most vulnerability instances in the application server, which is rectified to the database management component when accounting only entries from NVD.

4.4 Data Inconsistencies in CVSS, Impact Element and Weakness Type

The inconsistencies between different vulnerability report sources are reflected in weakness categorisation, impact evaluation, and vulnerability severity. We examined the CVSS version 2 base-scores of our investigated systems to check if inconsistent vulnerabilities affect the average severity of these investigated systems. As depicted in Figure 3, the customer management system and IoT system have almost similar vulnerability severity score distributions no matter which cybersecurity data source is used. In contrast, vulnerabilities from NVD entries contribute more vulnerabilities with higher severity scores (equal to or larger than 7) to Data Centre, Network, and Control System. This is particularly true for the communication network related vulnerability instances, whereby a large ratio of NVD vulnerability entries has a CVSS V2 base score between 7.0 and 8.0. Another interesting observation is that, vulnerabilities found in NVD contribute on average a lower CVSS score in terms of the OT system when compared to the instances found in the vendor sites.

We measured the influence of inconsistent vulnerabilities on exploitability and impact levels of the whole investigated system, as shown in Figure 4. Only accounting for NVD entries, these vulnerability instances may lead cybersecurity analysers to pay closer attention to threats with local path based attacks, low access complexity, as well as no authentication requirement. Meanwhile, the analysis result suggests that our investigated system is suffered from a higher impact on confidentiality, integrity and availability.

We then correlated the extracted vulnerabilities with weakness enumerations *CWE*[16] to explore the major weaknesses of each investigated system. As shown in Table 4, the top 3 weaknesses with the highest occurrence frequencies in all the investigated subsystems are partially changed or revised with rankings, except for the control system. For instance, the most frequently appeared network weaknesses when only accounting vendor entries are (i) exposure of sensitive information to an unauthorised actor, and (ii) incorrect permission assignment for the critical resource. Among these two network weaknesses, the first weakness occurs due to direct or indirect insertion of sensitive information such as system environment, network status and configuration, intellectual property, private customer records, etc., into resources that are accessible to unauthorised actors. This information leak may be exploited with attacks like excavation and fingerprinting and may lead to loss of

[16] https://cwe.mitre.org/index.html

IoT System	Component Number	Consider NVD Only		Consider Vendor Entries Only	
		Vulnerability	Component with The Most Vulnerabilities	Vulnerability	Component with The Most Vulnerabilities
Operating System	3	21	Directory Component	12	Database Toolset Component
Access Control (AC) Server	16	48	Anti-malware Component	28	Runtime Library Component
Encryption Key-store (EKM) Server	36	75	SDK Component	55	Runtime Library Component
Database (DB) Server	40	66	Database Management Component	46	Runtime Library Component
Application (APP) Server	43	74	Database Management Component	46	Runtime Library Component
Customer Management System	Component Number	Consider NVD Only		Consider Vendor Entries Only	
		Vulnerability	Component with The Most Vulnerabilities	Vulnerability	Component with The Most Vulnerabilities
Operating System	4	20	Remote Desktop Component	19	Remote Desktop Component
Customer Management (CM) Server	64	253	Office Application Component	232	Office Application Component
Database (DB) Server	52	68	SDK Component	48	SDK Component
Web (WEB) Server	17	44	Runtime Library Component	17	Runtime Library Component
Control System	Component Number	Consider NVD Only		Consider Vendor Entries Only	
		Vulnerability	Component with The Most Vulnerabilities	Vulnerability	Component with The Most Vulnerabilities
Operating System	4	12	Directory Component	8	Directory Component
Application (APP) Server	23	65	Database Management Component	45	Runtime Library Component
Other Systems	Component Number	Consider NVD Only		Consider Vendor Entries Only	
		Vulnerability	Component with The Most Vulnerabilities	Vulnerability	Component with The Most Vulnerabilities
Data Center	8	19	Hypervisor Component	11	Hypervisor Component
Network	7	75	Network Operating Component	11	Firewall Component
OT	7	5	Open Platform Communications Component	25	Protection and Control Component

Table 3: Retrieved Vulnerability Instances Considering Different Data Sources

confidentiality. Possible mitigations against this weakness include encryption and password-protection of sensitive data and appropriate compartmentalisation that reinforces privilege separation functionality. The second network weakness, or incorrect permission assignment for critical resource, may impact confidentiality and integrity of sensitive properties once successfully exploited by attacks like signing malicious code. The second network weakness can be remediated by environment hardening. The remaining network weaknesses are singular cases and are not listed in Table 4. Taking NVD as the only data source, then the major network weaknesses are revised as (i) improper input validation, (ii) improper neutralisation of special elements used in an OS command, and (iii) uncontrolled resource consumption. These ambiguous weaknesses may address more attention towards protecting the system against code execution, DoS, bypass protection mechanism, and other similar cyber threats, while resulting in less budget in mitigating the existing weaknesses.

4.5 Case Study Evaluation

We interviewed five cybersecurity experts to collect their feedback on the applicability and usefulness of our data inconsistency analysis. Among these five experts, three of them are employed in the investigated organisation in Sweden to ensure the confidentiality of the studied system and vulnerability data. These three experts cover the roles of IT administrators, IT security technicians and OT security operator. They hence have in-depth knowledge of the system status. The other two interviewees work as cybersecurity researcher and industrial cybersecurity consultant in Sweden, separately. These two interviewees are not involved in this case study. The feedback from these five interviewees is summarised below.

A significant number of vulnerability reports are found using public vulnerability repositories, especially for embedded software in the customer management system and control system. These retrieved vulnerability instances deliver valuable references and a broad picture of the vulnerable level of the whole system. Vulnerability analysis in the perspective of severity scores and major threat categories also guide patching prioritisation. Still, some known vulnerabilities in the OT system cannot be found using online cybersecurity repositories and are not published by the vendor. The lack of OT vulnerability records also indicates the incompleteness of online cybersecurity data sources.

The vulnerability analysis results indicate some weaknesses that the three IT technicians are not aware of. One interesting example refers to the office application components that contribute a significant amount of vulnerability instances in the customer management system, as shown in Table 3. These vulnerability instances also have a high ratio of critical severity scores. Nevertheless, office application components are regarded as low criticality and may not address enough attention in the perspective of cybersecurity in this company. Another example is the runtime library components that contribute significant numbers of vulnerabilities and are challenging to maintain. According to the cybersecurity researcher, it is also surprising to know that the firewall component in the network has a high vulnerable level. Data inconsistencies are not uncommon for vulnerabilities' affected products and version range. These data inconsistencies bring a higher level of uncertainty in cybersecurity

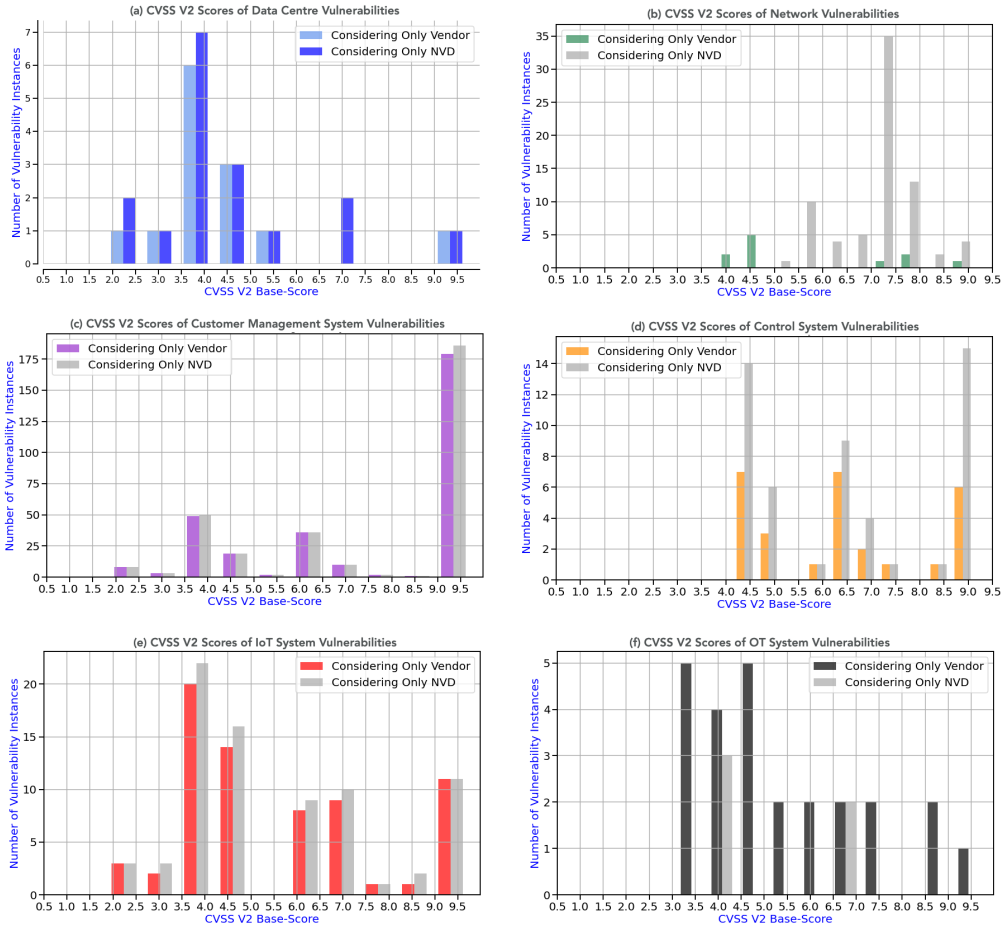

Figure 3: CVSS Version2 Scores of The Investigated Systems

decision making with amendments in the ratios of major weakness and threat types.

4.6 Discussion

4.6.1 Limitation. Query generation, vulnerabilities retrieval, and data inconsistencies evaluation were conducted in a semi-automated manner in this case study. To be specific, we generated query tags through both automatic terminology extraction and manual check. These query tags were matched against cybersecurity data sources to obtain related vulnerability instances. We also manually checked the inconsistent vulnerabilities to filter out instances that do not apply to our system. The whole process took around four weeks to complete, which may leave a gap between a vulnerability exploit occurrence and the deployment of an available patch. Meanwhile, relying on individual experts' knowledge could introduce recurrent costs, subjective evaluations and contradicting outcomes. Improving the level of automation supports a timely vulnerability-analysis lifecycle and allows mitigation to occur within the time interval that span the disclosure and patch of vulnerabilities. We did not investigate the root causes of the inconsistencies, i.e. which vulnerability instance was created first, when it was updated, when or if duplicates were detected, and so forth. Our study does however indicate that better workflows should be installed to synchronise and correct vulnerability instances in the public and vendor repositories.

4.6.2 Key Insights. Multiple cybersecurity reports and bulletins provide various perspective for vulnerability analysis. Yet, discrepant reports bring challenges to cybersecurity decision making in terms of patching prioritisation. Our case study results indicate that cybersecurity budget allocation may differ diversely by relying upon only NVD vulnerability entries or only vendor security bulletins. The systematic integration of vulnerability instances into a kind of data warehouse offers the opportunity to detect inconsistencies and refer them to the maintainers of the original sources. The case study also showed that the number of inconsistencies could largely differ depending on the type of software component. An approach that deploys trustworthiness data verification brings substantial benefits to vulnerability identification and management. Furthermore, vulnerabilities are reported in various standards, which increases the difficulty of vulnerability-related provisioning and sharing. Vulnerability reporting standardisation is

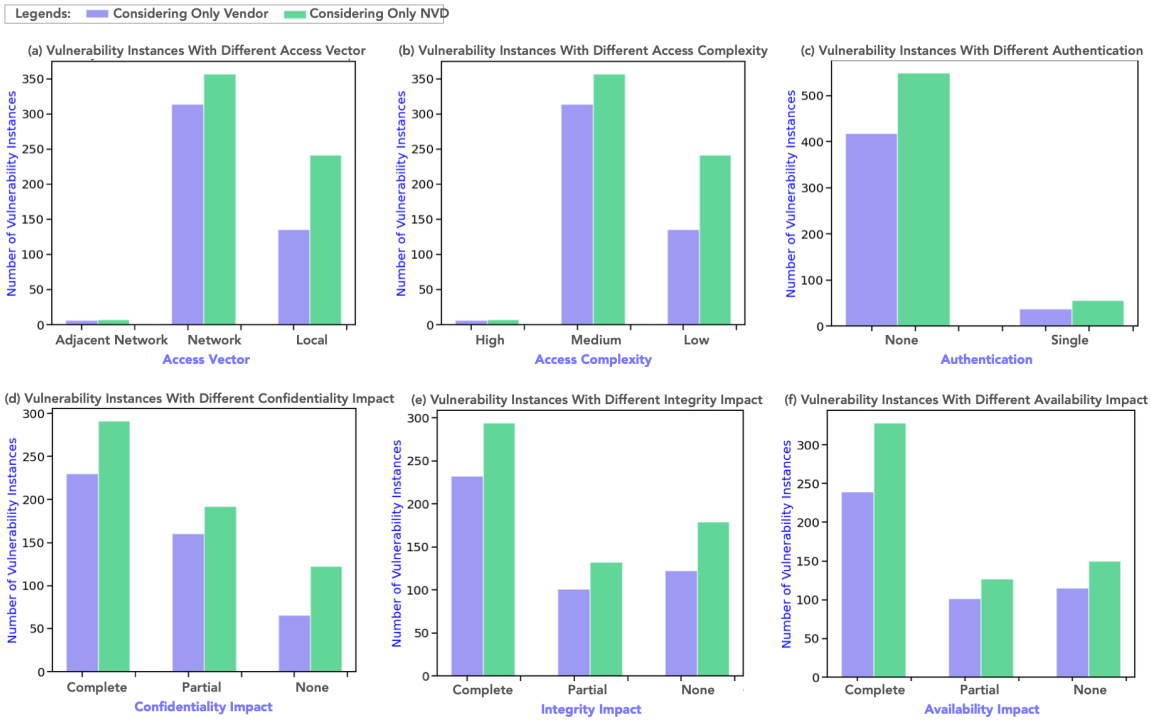

Figure 4: Exploitability and Impact of The Investigated Systems

Investigated System	Most Frequently Appeared Weaknesses Consider NVD Only	Most Frequently Appeared Weaknesses Consider Vendor Entries Only
DataCentre	1. Improper use of previously-freed memory. 2. Improper certificate validation. 3. Exposure of sensitive information to an unauthorised actor.	1. Improper certificate validation. 2. Insufficiently protected credentials. 3. Improper use of previously-freed memory.
Network	1. Improper input validation. 2. Improper neutralisation of special elements used in an OS command. 3. Uncontrolled resource consumption.	1. Exposure of sensitive information to an unauthorised actor. 2. Incorrect permission assignment for critical resource.
IoT System	1. Improper privilege management. 2. Improper permissions and access controls. 3. Improper input validation.	1. Improper privilege management. 2. Improper input validation. 3. Improper permissions and access controls.
Customer Management System	1. Improper restriction of operations within the bounds of a memory buffer. 2. Improper input validation. 3. Exposure of sensitive information to an unauthorised actor.	1. Improper restriction of operations within the bounds of a memory buffer. 2. Exposure of sensitive information to an unauthorised actor. 3. Improper input validation.
Control System	1. Improper neutralisation of input during web page generation. 2. Improper control of generation of code. 3. Improper permissions and access controls.	1. Improper neutralisation of input during web page generation. 2. Improper control of generation of code. 3. Improper permissions and access controls.
OT	1. Incorrect permission assignment for critical resource. 2. Improper privilege management.	1. Out-of-bounds Read. 2. Out-of-bounds Write. 3. Incorrect permission assignment for critical resource.

Table 4: Most Frequently Appeared Weaknesses

one way to enhance the quality of shared cyber threat intelligence (CTI).

5 CONCLUSION

Verifying the trustworthiness of vulnerability information is challenging but necessary before employing this information into cybersecurity decision making. In this paper, we evaluate the data inconsistencies of open-source vulnerability repositories in vulnerability assessment, particularly in the perspective of cybersecurity awareness enhancement. To do so, we carried out an empirical case study of the IT and OT system of a company in Sweden, mainly its data centre, network, control software, and the IoT segments. During this case study, we extracted a significant amount of vulnerability instances, based on which we reveal some unexpected weaknesses that decision-makers were not aware of before. We also observed that inconsistent vulnerabilities are retrieved from CVE, NVD and vendor websites using the exact query keywords. We then systematically measured the influence of inconsistent vulnerabilities on exploitability and impact levels, and the major weakness types of the whole investigated system. The high rate of inconsistent vulnerabilities in the control system and the network components are only valid in this case study. The rate of data inconsistency may differ in other organisations. Still, specific sub-systems can render a high percentage of data inconsistency.

To further enhance public cybersecurity data sources, we plan to explore data reliability validation automation. Vulnerability repositories of trusted vendors could be incorporated into a cross-linked local vulnerability database to detect inconsistencies automatically. Other cybersecurity data sources like security blogs where vulnerabilities are first identified also provide valuable references to be integrated. Natural language processing [18] and text mining techniques [12] can be employed to automate correlation between multiple cybersecurity repositories and correction of inconsistent vulnerability information.

REFERENCES

[1] Afsah Anwar, Ahmed Abusnaina, Songqing Chen, Frank Li, and David Mohaisen. 2020. Cleaning the NVD: Comprehensive Quality Assessment, Improvements, and Analyses. *arXiv preprint arXiv:2006.15074* (2020).

[2] Terje Aven, Yakov Ben-Haim, H Boje Andersen, Tony Cox, Enrique López Droguett, Michael Greenberg, Seth Guikema, Wolfgang Kröger, Ortwin Renn, Kimberly M Thompson, and others. 2018. Society for risk analysis glossary. *Society for Risk Analysis, August* (2018).

[3] Oscar Chaparro, Jing Lu, Fiorella Zampetti, Laura Moreno, Massimiliano Di Penta, Andrian Marcus, Gabriele Bavota, and Vincent Ng. 2017. Detecting missing information in bug descriptions. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 396–407.

[4] Ying Dong, Wenbo Guo, Yueqi Chen, Xinyu Xing, Yuqing Zhang, and Gang Wang. 2019. Towards the detection of inconsistencies in public security vulnerability reports. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 869–885.

[5] Sadegh Farhang, Mehmet Bahadir Kirdan, Aron Laszka, and Jens Grossklags. 2020. An Empirical Study of Android Security Bulletins in Different Vendors. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, Yennun Huang, Irwin King, Tie-Yan Liu, and Maarten van Steen (Eds.). ACM / IW3C2, 3063–3069. DOI: http://dx.doi.org/10.1145/3366423.3380078

[6] Allen D Householder, Garret Wassermann, Art Manion, and Chris King. 2017. *The cert guide to coordinated vulnerability disclosure.* Technical Report. Carnegie-Mellon Univ Pittsburgh Pa Pittsburgh United States.

[7] Matthias Jarke, Manfred A Jeusfeld, Christoph Quix, and Panos Vassiliadis. 1999. Architecture and quality in data warehouses: An extended repository approach. *Information Systems* 24, 3 (1999), 229–253.

[8] Yuning Jiang, Yacine Atif, and Jianguo Ding. 2020. Cyber-Physical Systems Security Based on a Cross-Linked and Correlated Vulnerability Database. In *Critical Information Infrastructures Security*, Simin Nadjm-Tehrani (Ed.). Springer International Publishing, Cham, 71–82.

[9] Hyeonseong Jo, Jinwoo Kim, Phillip Porras, Vinod Yegneswaran, and Seungwon Shin. 2020. GapFinder: Finding Inconsistency of Security Information From Unstructured Text. *IEEE Transactions on Information Forensics and Security* 16

(2020), 86–99.
[10] Xiaojing Liao, Kan Yuan, XiaoFeng Wang, Zhou Li, Luyi Xing, and Raheem Beyah. 2016. Acing the ioc game: Toward automatic discovery and analysis of open-source cyber threat intelligence. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 755–766.
[11] David Loshin. 2010. *Master data management*. Morgan Kaufmann.
[12] Syed Shariyar Murtaza, Wael Khreich, Abdelwahab Hamou-Lhadj, and Ayse Basar Bener. 2016. Mining trends and patterns of software vulnerabilities. *Journal of Systems and Software* 117 (2016), 218–228.
[13] Antonio Nappa, Richard Johnson, Leyla Bilge, Juan Caballero, and Tudor Dumitras. 2015. The attack of the clones: A study of the impact of shared code on vulnerability patching. In *2015 IEEE symposium on security and privacy*. IEEE, 692–708.
[14] Viet Hung Nguyen and Fabio Massacci. 2013. The (un) reliability of nvd vulnerable versions data: An empirical experiment on google chrome vulnerabilities. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. 493–498.
[15] MingJian Tang, Mamoun Alazab, and Yuxiu Luo. 2017. Big data for cybersecurity: Vulnerability disclosure trends and dependencies. *IEEE Transactions on Big Data* 5, 3 (2017), 317–329.
[16] Zhilin Yang, Ruslan Salakhutdinov, and William W Cohen. 2017. Transfer learning for sequence tagging with hierarchical recurrent networks. *arXiv preprint arXiv:1703.06345* (2017).
[17] Xiaoxin Yin, Jiawei Han, and S Yu Philip. 2008. Truth discovery with multiple conflicting information providers on the web. *IEEE Transactions on Knowledge and Data Engineering* 20, 6 (2008), 796–808.
[18] Ziyun Zhu and Tudor Dumitraş. 2016. Featuresmith: Automatically engineering features for malware detection by mining the security literature. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 767–778.
[19] Ziyun Zhu and Tudor Dumitras. 2018. Chainsmith: Automatically learning the semantics of malicious campaigns by mining threat intelligence reports. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 458–472.