Cards and Stacks:
A Dynamic, Asset-Centric and Machine Learning-Enhanced Approach to Cyber Vulnerability Management

by Adham Shaaban

B.A. in Computer Science, May 1998, University of San Diego
M.S. in Information Systems, June 2017, San Diego State University

A Praxis submitted to

The Faculty of
The School of Engineering and Applied Science
of The George Washington University
in partial fulfillment of the requirements
for the degree of Doctor of Engineering

May 19, 2024

Praxis directed by

Domingo Rivera
Professorial Lecturer of Engineering and Applied Science

The School of Engineering and Applied Science of The George Washington University certifies that Adham Shaaban has passed the Final Examination for the degree of Doctor of Engineering as of April 2, 2024. This is the final and approved form of the Praxis.

Cards and Stacks:
A Dynamic, Asset-Centric and Machine Learning-Enhanced Approach to Cyber Vulnerability Management

Adham Shaaban

Praxis Research Committee:

Domingo Rivera, Professorial Lecturer of Engineering and Applied Science, Praxis Director

Linda Rodriguez, Professorial Lecturer of Engineering and Applied Science, Committee Member

Muhammad Islam, Professorial Lecturer of Engineering and Applied Science, Committee Chair

© Copyright 2024 by Adham Shaaban
All rights reserved.

Dedication

I wish to dedicate this praxis to my wife Lolbe and children Kenzie and Kareem. Please forgive me for the many family nights and events missed while working to complete this program and praxis. I could not have done it without your unconditional love and support, and I love you all dearly. And to my late parents who taught me to never stop pursuing knowledge or chasing my dreams.

Acknowledgements

I would like to acknowledge my wife and kids for their unconditional love and support that made this praxis possible, my late parents for their guidance from beyond, my good friend and amazing editor Mr. Robert Tirsbier for his review, my CEO Carlos Persichetti for his professional support, my advisor Dr. Domingo Rivera for his guidance and support throughout the praxis research phase, Dr. Muhammad Islam, chairman of the praxis defense committee, Dr. Linda Rodriguez, member of the committee, and the entire faculty and staff of the Doctor of Engineering in Cybersecurity Analytics program at The George Washington University, who provided the foundation for the successful completion of this praxis research.

Abstract of Praxis

Vulnerability management is a critical component of defensive cyber operations. However, there are several weaknesses with the current approaches to vulnerability management. First, it has become increasingly difficult to map known vulnerabilities to an organization's affected systems. The Common Vulnerabilities and Exposures (CVE) program has logged over 200,000 vulnerabilities since inception and that number is projected to continue to rise exponentially, making it challenging for records to contain complete or timely information. Second, organizations often do not know exactly what components make up their information systems and particularly the software products within those systems. This leads to two common questions related to vulnerability management: "are we affected?" and "where are we affected?" Third, with only an estimated 2% of reported vulnerabilities actually exploited by malicious actors, the current approach of scanning for every single vulnerability fails to prioritize critical ones and often produces a high number of irrelevant results. This "noise" often overwhelms human cyber analysts charged with triaging the findings and leads to missing the more important “signals”. To help address these challenges, the concept of a Software Bill of Materials (SBOM) has recently emerged as an important topic in cybersecurity in general and vulnerability management in particular. This concept has taken even more urgency after the President of the United States issued Executive Order 14028 on Improving the Nation's Cybersecurity, which mandates that software developers prepare and transmit SBOMs to vendors, customers and users. For the purposes of our research, we broadly define an IT asset as any IT resource that provides value to an organization’s business

operations. We introduce the concept of an Asset Bill of Materials (ABOM) – an extension and generalization of the SBOM concept – as a method to describe all components of an IT asset. We present a novel "Cards and Stacks" model of an ABOM and propose a dynamic, asset-centric, machine learning-enhanced approach to vulnerability management based on the ABOM concept. We then use CVE data along with a sample ABOM to demonstrate how our approach can assist with the discovery of relevant vulnerability information about assets in a sample IT environment. Finally, we propose areas of future research on extending and enhancing our ABOM approach.

Table of Contents

Dedication iii
Acknowledgements iv
Abstract of Praxis v
List of Acronyms 13
Glossary of Terms 14
Chapter 1 - Introduction 16
1.1 Background 16
1.1.1 Role of Vulnerability Management in Cybersecurity 16
1.1.2 Asset Management 19
1.1.3 Common Vulnerabilities and Exposures (CVE) 20
1.1.4 Common Platform Enumeration (CPE) 23
1.1.5 National Vulnerability Database 27
1.1.6 Common Weakness Enumeration (CWE) 28
1.1.7 Software Bill of Materials (SBOM) 29
1.2 Research Motivation 29
1.3 Problem Statement 30
1.4 Thesis Statement 31
1.5 Research Objectives 31
1.6 Research Questions and Hypothesis 32
1.7 Scope of Research 32
1.8 Research Limitations 33
1.9 Organization of Praxis 33
Chapter 2 - Literature Review 35
2.1 Introduction 35
2.2 Asset Management 35
2.3 Vulnerability Management and CVE Issues 42
2.4 Software Bill of Materials 46
2.5 Summary and Conclusion 49

Chapter 3 - Methodology **51**
3.1 Introduction 51
3.2 The Asset Bill of Materials (ABOM) Concept 52
3.2.1 Introduction....52
3.2.2 SBOM Formats and Standards....53
3.3 The Cards and Stacks Model 57
3.3.1 JavaScript Object Notation (JSON) Schema....59
3.3.2 CSV Schema....61
3.3.3 Sample Asset Stack....62
3.4 Python and development tools 66
3.5 Data gathering and dataset building 67
3.5.1 NVD API....67
3.5.2 NVDLib Python Library....69
3.5.3 Primary Dataset....71
3.6 Data labelling and clean-up 72
3.7 Asset Stack vs. Individual Software Tool CVE Description Search 73
3.8 Text Analysis Approach 74
3.8.1 Term Frequency – Inverse Document Frequency....75
3.9 Machine Learning Model description 77
3.9.1 Decision Tree....77
3.9.2 Random Forest....79
3.9.3 Applying the Model to our CVE Descriptions Dataset....81
Chapter 4 - Results **83**
4.1 Introduction 83
4.2 CVE Analysis 83
4.3 Impact of asset-centric approach on CVE discoverability 85
4.4 Results of using ML text analysis for predicting CVE relevance 86
Chapter 5 - Discussion and Conclusion **90**
5.1 Discussion 90

5.2 Conclusions 91

5.3 Contributions to Body of Knowledge 93

5.4 Recommendations for Future Research 93

Appendix A – ABOM-based Vulnerability Scanner Proof of Concept 96

References 103

List of Figures

Figure 1. Record of CVE-2023-34362 from CVE.org in the JSON 5.0 format 23

Figure 2. ABNF for Formatted String Binding..... 26

Figure 3. CPE Match String..... 27

Figure 4. CPE Match String Range..... 27

Figure 5. Example of IIOT System Structure and XML representation..... 39

Figure 6. Example of deprecated CPE entry due to an identifier name correction 44

Figure 7. Semantically equivalent CPEs with different WFN attributes between the official CPE dictionary (first entry) and the CVE records (second entry)..... 45

Figure 8. Flowchart of research steps 52

Figure 9. Components of an SPDX document..... 53

Figure 10. CycloneDX SBOM components 55

Figure 11. Example of a SWID Primary Tag 57

Figure 12. Visual representation of a device asset as a stack of cards and stacks 59

Figure 13. A stack containing a single card. 60

Figure 14. A stack containing one card and one nested stack with two cards. 61

Figure 15. JSON representation of the Mozilla Firefox browser stack 65

Figure 16. CSV representation of the Mozilla Firebox browser stack 66

Figure 17. Sample NVD API Call 68

Figure 18. Looping API calls to retrive a large number of records from the NVD..... 69

Figure 19. Example use of the searchCVE() API method to retrieve CVE records..... 70

Figure 20. Use of custom parameters with the searchCVE() API method 70

Figure 21. Sample CSV record from the primary dataset..... 71

Figure 22. Code for extracting relevant CVEs..... 72

Figure 23. Searching for CVE records using asset names only..... 74

Figure 24. Searching for CVE records using asset AND sub-component names..... 74

Figure 25. Sample call of the TfidfVectorizer() method from the scikit-learn library..... 76

Figure 26. Decision Tree Structure..... 78

Figure 27. Decision tree to determine tastiness of a papya fruit..... 79

Figure 28. Sample Random Forest..... 80

Figure 29. Creating a 70/30 split training and testing datasets..... 81

Figure 30. Creating the Random Forest Classifier in scikit-learn 82

Figure 31. Generating the predictions using the text classifier..... 82

Figure 32. Generating a confusion matrix to evaluate the performance of our model..... 82

Figure 33. Number of CVEs published per quarter. 85

Figure 34. CVE search results for asset stacks (assets with components) vs. assets only 86

Figure 35. Visualization of confusion matrix of balanced dataset using Random Forest 87

Figure 36. Sample stack input file for Python proof of concept..... 98

Figure 37. Python source code for proof of concept..... 102

Figure 38. Console output from a sample run of the proof of concept..... 102

List of Tables

Table 1. Fields of the CSV format representation of the cards and stacks 62

Table 2. List of software comprising the sample asset stack 63

Table 3. Number of CVEs published per quarter. 84

Table 4. Confusion Matrix of balanced dataset using Random Forest 87

Table 5. Classification report for random forest model 88

List of Acronyms

ABOM	Asset Bill of Materials
API	Application Programming Interface
CISA	Cybersecurity and Infrastructure Security Agency
CMDB	Configuration Management Database
CNA	CVE Numbering Authority
CPE	Common Platform Enumeration
CSV	Comma Separated Values
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
DCO	Defensive Cyber Operations
EO	Executive Order
IOT	Internet of Things
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
ML	Machine Learning
NIST	National Institute of Standards and Technology
NTIA	National Telecommunications and Information Administration
NVD	National Vulnerability Database
SBOM	Software Bill of Materials
SCA	Software Component Analysis
SPDX	Software Package Data Exchange
SWID	Software Identification Tags
TF-IDF	Term Frequency-Inverse Document Frequency
WFN	Well-Formed CPE Names

Glossary of Terms

Asset	An Information Technology resource that provides value to the business operations of an organization such as hardware, software, data, cloud resources, mobile devices, Internet-of-Things (IOT) devices and virtual computing resources.
Asset Bill of Materials (ABOM)	A broadened generalization of the Software Bill of Materials (SBOM) concept that allows for the representation of all of the various components and sub-components that constitute a modern IT asset.
Common Platform Enumeration (CPE)	A standardized method created by the National Institute for Standards and Technology (NIST) to identify and describe classes of enterprise computing assets (such as applications, operating systems and hardware devices) and facilitate the application and verification of IT management policies related to such assets.
Common Vulnerabilities and Exposures (CVE)	A program run by MITRE Corporation that has become the de facto standard for the public sharing of technical vulnerability information. It aims to identify, define and catalog publicly disclosed cybersecurity vulnerabilities by assigning a unique CVE identifier and creating a record for each reported vulnerability.
Common Weakness Enumeration (CWE)	A community-developed dictionary (maintained by the MITRE corporation) of the most common types of conditions (i.e. bugs or design flaws) in software, firmware, hardware or service components that may contribute to the introduction of vulnerabilities.
JavaScript Object Notation (JSON)	A lightweight data-interchange format based on a subset of the JavaScript programming language. It's commonly used to describe data as it is easy for machines to parse and generate while also being easy for humans to read and understand.
Defensive Cyber Operations (DCO)	Cybersecurity operations (active or passive) that aim to protect networks, computers, data and all designated cyber systems and resources.
Machine Learning (ML)	A subfield of Artificial Intelligence that studies the design and use of algorithms trained on data sets that can generalize to new data and allow machines to perform certain tasks without explicit human instructions.
National Vulnerability Database (NVD)	A national database maintained by the United States National Institute of Standards and Technology (NIST) that collects, analyzes, enriches and stores the vulnerability records published through the CVE program. It is described by NIST as "the official United States government repository of standards-based vulnerability management

	data represented using the Security Content Automation Protocol (SCAP).” (National Institute of Standards and Technology, 2023)
Software Bill of Materials (SBOM)	A Software Bill of Materials (SBOM) is defined by the United States Cybersecurity and Infrastructure Security Agency (CISA) as “a nested inventory, a list of ingredients that make up software components” that serves as “a key building block in software security and software supply chain management”. Work on SBOM standards has been a collaborative community effort driven by the National Telecommunications and Information Administration (NTIA) multi-stakeholder process since 2018.
Software Component Analysis (SCA)	Software Component Analysis (SCA) is the process of identifying potential areas of risk from using third-party and open-source software components in a software system. It is a core function within Cyber Supply Chain Risk Management frameworks.
Software Package Data Exchange (SPDX)	An open standard maintained by the community driven SPDX Project (part of the Linux Foundation). It is commonly used in software bill-of-materials for identifying the components, licenses, copyrights, and security metadata relating to software. It was originally created to help streamline license compliance but later expanded for additional use cases.
Term Frequency-Inverse Document Frequency (TF-IDF)	Term Frequency-Inverse Document Frequency (TF-IDF) is a statistical measure of how relevant a word is to a document in a collection of documents. It is calculated by multiplying two component metrics: how many times a word appears in a document and the inverse document frequency of the word access all documents in the collection. It increases proportionally by the number of times a word appears in a document offset by the number of documents that contain the word. Thus, words that are common in every document will rank lower while words that are unique to a document will rank higher indicating their higher relevance to the document. It is considered a very useful technique for scoring words in machine learning algorithms related to natural language processing (NLP)

Chapter 1 - Introduction

1.1 Background

1.1.1 Role of Vulnerability Management in Cybersecurity

The Cybersecurity and Infrastructure Security Agency (CISA) is the United States Federal Agency tasked with protecting federal information systems and coordinating the security and resilience of critical US infrastructure. CISA defines Cybersecurity as "the art of protecting networks, devices, and data from unauthorized access or criminal use and the practice of ensuring confidentiality, integrity, and availability of information" (CISA, 2021). Cybersecurity combines science, technology, tactics, techniques, procedures and the common body of knowledge developed to protect our increasingly interconnected modern digital society. It plays a critical part in securing the supply chains that provide us with food, supplies and almost all the physical goods we purchase and use in our daily lives. And while the Cybersecurity discipline encompasses both offensive and defensive elements, the focus of most cybersecurity professionals (and this research paper) is on the defensive cyber operations (DCO) that aim to stop increasingly sophisticated and frequent cyber-attacks that target every aspect of our digital lives, from personal communications to financial transactions and from healthcare to critical infrastructure (Kelley, 2023).

In 2021, the number of worldwide ransomware attempts rose to 623.5 million a nearly 104% increase over the 304.64 million attempts made in 2020. Meanwhile, the number of annual worldwide attacks across all malware categories reached 5.5 billion attacks in 2022 (Petrosyan, 2023). The economic impact of these cyber-attacks is staggering. In 2021, the global cost of cybercrime was estimated to be $6.9 trillion with

this number expected to rise to $10.5 trillion by 2025. In 2022, the global average data breach cost $4.35 million with the healthcare industry accounting for the costliest breaches at an average cost of $10.1 million per incident (Kolesnikov, 2023).

Malicious cyber actors aim to exploit vulnerabilities in the information systems they target. Vulnerability management is the process of identifying, analyzing, tracking and ultimately fixing or mitigating vulnerabilities in an organization's critical assets (Muniz, McIntyre, & AlFardan, 2015). Vulnerabilities may exist in people, processes, or technologies and technical vulnerability types may vary from software bugs to misconfiguration and hardware issues. Identifying and mitigating vulnerabilities is an essential requisite for stopping attackers from exploiting them and it's often a critical security control in common cybersecurity frameworks such as ISO 27001 published by the International Organization for Standardization (ISO) and NIST Special Publication 800-53 published by the National Institute for Science and Technology (NIST).

The ISO 27001 standard focuses on creating an information security management system that covers the confidentiality, integrity and availability of information as part of an organization's risk management process (International Organization for Standardization, 2022a). The ISO 27001 security control A.12.6.1 (titled "Control of Technical Vulnerabilities") identifies three primary tenants of a vulnerabilities management program:

1. **Timely identification of vulnerabilities**: the detection and remediation of vulnerabilities in an information system must happen in a timely fashion.

2. **Assessment of exposure to vulnerabilities:** Assigning severity levels to identified vulnerabilities will allow an organization to identify the most critical ones and prioritize remediation actions accordingly.
3. **Provide proper counter measures to identified risks:** After the identification of the most critical vulnerabilities, an organization should propose and take adequate countermeasures to address the identified risks.

In addition to the ISO 27001 standard, ISO also publishes the ISO 27002 document which provides specific implementation guidance for the reference set of information security controls. ISO 27002 includes a set of best practices for implementing the vulnerability management control A.12.6.1 (RiskOptics, 2021) including:

- **Inventory of assets**: an organization must have knowledge of its information assets including software manufactures, versions, deployment locations and system ownership.
- **Establishing roles and responsibilities**: defining the roles and responsibilities of the human resources involved in the vulnerability management process.
- **Timeline for reaction**: defining the timeline to react to the discovery of technical vulnerabilities.
- **Audit log**: maintain an audit log for the process.
- **Aligning the system with incident management**: aligning the vulnerability management process with related incident response and reporting activities within the organization

- **Continual improvement through correction action and preventive action (CAPA):** ensuring that the vulnerability management security controls continue to function and adapt to new and emerging threats.

In addition to industry standards and best practices, certain government and industry-specific rules and regulations (such as PCI DSS for the financial industry, HIPPA for the healthcare industry and DFARS for the defense industrial base) require organizations in the corresponding industries to have vulnerability management systems in place.

Vulnerability Management is therefore a critical component of any defensive cybersecurity program, and it starts with identifying the organizational assets that could be vulnerable.

1.1.2 Asset Management

For the purposes of our research, we broadly define an "asset" as any IT resource that provides value to the business operations of an organization. This definition allows an asset to encompass a wide range of components including hardware, software, data, cloud resources, mobile devices, Internet-of-Things (IOT) devices and virtual computing resources. Asset Management is the process by which an organization identifies, documents, and tracks its valuable assets. And while, in many organizations, asset management is not entirely an information security function, it is nonetheless critical to the overall security posture of an organization (Brotherston & Berlin, 2017). ISO 27002, for example, identifies asset management as a critical component of an effective vulnerability management process (International Organization for Standardization, 2022b)

Without proper asset management, a vulnerability management program can only provide abstract information that would lack operational context and fail to answer the "how are we affected?" question. For example, the program may identify a vulnerability in a common software or hardware product. However, this information can only be useful when paired with an asset management system that enables security personnel to identify the specific instances of such products being used within the organization and determine the true impact of the vulnerability.

1.1.3 Common Vulnerabilities and Exposures (CVE)

The Common Vulnerabilities and Exposures (CVE) program run by MITRE Corporation has become the unofficial standard for the public sharing of technical vulnerability information. Technology vendors and the global cybersecurity research community use the program to publicly identify, define and catalog newly discovered vulnerabilities in software and hardware products (MITRE, 2023b). Each vulnerability record contains a unique identifier (ID), the vulnerability description, and a references field for URLs and other information. The CVE unique identifier has the form of CVE-YYYY-NNNN. It starts with the CVE prefix followed by the four digits of the year the vulnerability was made public (not the year when the vulnerability was actually introduced into the affected software or hardware product, a date that is often much earlier). The identifier concludes with the serial number of the vulnerability within that year. For example, the infamous vulnerability in the Apache Log4j library had the following unique ID: CVE-2021-44228. CVE IDs are issued by CVE Numbering Authorities or CNAs. The MITRE Corporation has traditionally been the primary CNA

but as the CVE and CNA programs evolved, MITRE categorized CNAs into 4 categories (MITRE, 2023a) :

1. **Sub CNA**: most common level of CNA. Each CNA assigns CVE IDs for vulnerabilities in their own products.
2. **Root**: manages a group of sub-CNAs within a given domain or community of interest.
3. **Top-Level Root**: special type of root that oversees their CNA hierarchy,
4. **CNA of Last Resort**: a CNA to cover any gaps for vulnerabilities not covered by Sub-CNAs within a root's hierarchy.

The CVE description field is a free text field that does not have a standard format. It's only advised that the field contain details about the vulnerability including the vulnerable software and impact of exploiting the vulnerability. This field may also contain a standard text description such as:

*"** RESERVED ** This candidate has been reserved by an organization or individual that will use it when announcing a new security problem. When the candidate has been publicized, the details for this candidate will be provided."*

Vendors are allowed to reserve IDs in blocks and this standard description denotes a CVE record has been reserved by a CNA or issued to a vendor that has not used it yet (Wikipedia contributors, 2023).

Figure 1 shows the official record for CVE-2023-34362 in the JSON 5.0 format as published on the CVE website. CVE-2023-34362 documents a vulnerability discovered in the Progress MOVEIt File Transfer software and highlights the importance of the CVE

program. Described as “the biggest hack of 2023”, CVE-2023-34362 has impacted over 1000 organizations and is estimated to have resulted in the unauthorized disclosure of the private information of over 60 million users including the protected health information of 11 million customers of a single organization (Page, 2023)

```
{
    "dataType": "CVE_RECORD",
    "dataVersion": "5.0",
    "cveMetadata": {
        "state": "PUBLISHED",
        "cveId": "CVE-2023-34362",
        "assignerOrgId": "8254265b-2729-46b6-b9e3-3dfca2d5bfca",
        "assignerShortName": "mitre",
        "dateUpdated": "2023-06-23T00:00:00",
        "dateReserved": "2023-06-02T00:00:00",
        "datePublished": "2023-06-02T00:00:00"
    },
    "containers": {
        "cna": {
            "providerMetadata": {
                "orgId": "8254265b-2729-46b6-b9e3-3dfca2d5bfca",
                "shortName": "mitre",
                "dateUpdated": "2023-06-23T00:00:00"
            },
            "descriptions": [
                {
                    "lang": "en",
                    "value": "In Progress MOVEit Transfer before 2021.0.6 (13.0.6), 2021.1.4 (13.1.4), 2022.0.4
(14.0.4), 2022.1.5 (14.1.5), and 2023.0.1 (15.0.1), a SQL injection vulnerability has been found in the MOVEit Transfer
web application that could allow an unauthenticated attacker to gain access to MOVEit Transfer's database. Depending on
the database engine being used (MySQL, Microsoft SQL Server, or Azure SQL), an attacker may be able to infer
information about the structure and contents of the database, and execute SQL statements that alter or delete database
elements. NOTE: this is exploited in the wild in May and June 2023; exploitation of unpatched systems can occur via
HTTP or HTTPS. All versions (e.g., 2020.0 and 2019x) before the five explicitly mentioned versions are affected,
including older unsupported versions."
                }
            ],
            "affected": [
                {
                    "vendor": "n/a",
                    "product": "n/a",
                    "versions": [
                        {
                            "version": "n/a",
                            "status": "affected"
                        }
                    ]
                }
            ],
            "references": [
                {
                    "url": "https://community.progress.com/s/article/MOVEit-Transfer-Critical-Vulnerability-31May2023"
                },
                {
                    "url": "http://packetstormsecurity.com/files/172883/MOVEit-Transfer-SQL-Injection-Remote-Code-
Execution.html"
                },
                {
                    "url": "http://packetstormsecurity.com/files/173110/MOVEit-SQL-Injection.html"
                }
            ],
            "problemTypes": [
                {
                    "descriptions": [
                        {
                            "type": "text",
                            "lang": "en",
                            "description": "n/a"
                        }
                    ]
                }
            ]
        }
    }
}
```


Figure 1. Record of CVE-2023-34362 from CVE.org in the JSON 5.0 format

1.1.4 Common Platform Enumeration (CPE)

The Common Platform Enumeration (CPE) is a standardized method created by the National Institute for Standards and Technology (NIST) to identify and describe classes of enterprise computing assets such as applications, operating systems and

hardware devices. The standard facilitates the application and verification of IT management policies related to such assets. These IT policies may include vulnerability management, configuration management and remediation policies. The standard also enables the use of automated tools for the identification of products and the sharing of information about those products (Cheikes, Cheikes, Kent, & Waltermire, 2011).

CPE consists of several modular specifications that can be combined together to serve different functions. The specification most pertinent to vulnerability management is the Naming Specification which defines a standardized method for naming IT product classes such as operating systems, software applications or hardware devices. Class names are known as Well-formed CPE Names (WFN) which are unordered sets of key-value pairs. For example, the following WFN represents any edition of the Microsoft Internet Explorer 8.0.6001 Beta browser:

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

WFNs have no prescribed order meaning the key-value pairs can be listed in any order and there is no specified hierarchical, set-theoretic or other relationship among the pairs (CSRC Content Editor, 2023).

WFNs are abstract logical constructs so the specification also defines procedures for binding these WFNs to machine-readable encodings to facilitate the exchange of this data among automated tools. One such binding provided in version 2.3 (the current version of the specification as of this writing) is known as “formatted string binding.” Formatted string binding represents the WFN for the same Internet Explorer browser as such:

cpe:2.3:a:microsoft:internet_explorer:8.0.6001:beta:*:*:*:*:*:*:*

The formatted string binding includes a number of special characters (Cheikes et al., 2011):

- The asterisk ("*") represents the logical value ANY.
- The hyphen ("-") represents the logical value NA.
- All other non-alphanumeric characters, if used, must be quoted by preceding them with a backslash).
- The special characters asterisk and question mark may appear without quoting, in which case they are open to special interpretation by other CPE specifications.

Figure 1 shows the Augmented Backus-Naur Form (ABNF) grammar — a standardized formal notation used across many Internet syntax specifications — for the formatted string binding.

```
formstring      = "cpe:2.3:" component-list

component-list  = part ":" vendor ":" product ":" version ":" update ":"
                  edition ":" lang ":" sw_edition ":" target_sw ":"
                  target_hw ":" other

part            = "h" / "o" / "a" / logical
vendor          = avstring
product         = avstring
version         = avstring
update          = avstring
edition         = avstring
lang            = LANGTAG / logical
sw_edition      = avstring
target_sw       = avstring
target_hw       = avstring
other           = avstring
avstring        = *1spec_chrs 1*( unreserved / quoted ) *1spec_chrs / logical
spec_chrs       = 1*spec1 / spec2
logical         = "*" / "-"
special         = spec1 / spec2
spec1           = "?"
spec2           = "*"
unreserved      = ALPHA / DIGIT / "-" / "." / "_"
quoted          = escape (escape / special / punc)
escape          = "\"
punc            = "!" / DQUOTE / "#" / "$" / "%" / "&" / "'" / "(" / ")" /
                  "+" / "," / "/" / ":" / ";" / "<" / "=" / ">" / "@" /
                  "[" / "]" / "^" / "`" / "{" / "|" / "}" / "~"
ALPHA           = %x41-5a / %x61-7A   ; a-z
DIGIT           = %x30-39  ; 0-9
DQUOTE          = %x22  ; double-quote
LANGTAG         = language ["-" region] ; cf. [RFC5646]
language        = 2*3ALPHA              ; shortest ISO 639 code
region          = 2ALPHA / 3DIGIT       ; ISO 3166-1 code or UN M.49 code
```

Figure 2. ABNF for Formatted String Binding

The CPE 2.3 Dictionary Specification defines a standardized method for creating repositories of CPE names and their associated metadata (known as CPE dictionaries). The official CPE Dictionary is hosted by NIST and is considered the authoritative repository of CPE identifiers (MITRE, 2013).

The NIST National Vulnerability Database (NVD) - described in detail later - uses the formatted string binding CPE format in its vulnerability detail pages (also known as “applicability statements”) to identify known affected software configurations

(National Institute of Standards and Technology, 2023). The applicability statements and their matching CPE names are not meant for human readers but rather to be primarily processed by automated tools. The NVD uses the concept of "match criteria" so that applicability statements can be shielded from changes in the official CPE dictionary and not require constant updates or maintenance. CPE Match criteria come in two forms:

1. **CPE Match Strings**: these are single CPE names that correlate to one or many CPE names in the official CPE dictionary). Figure 2 shows a sample CPE match string for the Microsoft Windows Server 2008 R2 SP1 operating system.

Figure 3. CPE Match String

2. **CPE Match String Ranges**: similar to CPE match strings but able to include both a start and end version assignments. Match String Ranges can represent <, >, <=, >= within a boundary. Figure 3 shows a sample CPE Match String Range for the phpMyAdmin web application up to and including version 4.9.0.1

Figure 4. CPE Match String Range

1.1.5 National Vulnerability Database

The vulnerability records published through the CVE program are collected, analyzed, enriched and stored in the National Vulnerability Database (NVD) maintained by the United States National Institute of Standards and Technology (NIST). The NVD

is the United States “government repository of standards based vulnerability management data represented using the Security Content Automation Protocol (SCAP)” (National Institute of Standards and Technology, 2023). The purpose of the NVD is to provide a complete set of information about vulnerabilities and a standard way to access that information that can enable automation of vulnerability management, security measurement and compliance.

1.1.6 Common Weakness Enumeration (CWE)

The Common Weakness Enumeration (CWE) is a community-developed dictionary of the most common types of conditions (i.e. bugs or design flaws) in software, firmware, hardware or service components that may contribute to the introduction of vulnerabilities. The list is maintained by the MITRE corporation and the current version 4.12 as of this writing contains 1395 entries. The CWE list is aimed at both software developers and security practitioners to raise awareness of - and ultimately prevent - the types of security vulnerabilities that have plagued the software and hardware industries. Available on their website (https://cwe.mitre.org), MITRE offers downloads of the entire list, a fully searchable online version, and narrower, more focused views such as the “CWE Top 25 Most Dangerous Software Weaknesses” and “CWE Most Important Hardware Weaknesses.” Additionally, MITRE provides external mappings between CWE content and related third-party security resources such as the popular Open Worldwide Application Security Project (OWASP) Top 10 list of most common web application security risks (MITRE Corporation, 2023).

1.1.7 Software Bill of Materials (SBOM)

Having automated tools and access to up-to-date vulnerability information is only half the battle. The dynamic and complex nature of modern information systems make it extremely difficult to track and inventory cyber assets and be able to identify all the components that make up such assets. This is particularly true for commercial software products that are akin to monolithic black boxes that provide little visibility into their constituent components. To address this specific problem, the concept of Software Component Analysis (SCA) and the Software Bill of Materials (SBOM) have started to gain prominence in recent years. In May 2021, President Joe Biden issued Executive Order (EO) 14028 on Improving the Nation's Cybersecurity. One of the key elements of the EO was the directive to improve the security of the software supply chain by establishing baseline security standards and requiring software developers and vendors to provide standardized Software Bill of Materials for their products (Biden, 2021). An SBOM is a formal, machine-readable inventory of software components and dependencies, information about the components, and their hierarchical relationships. The goal is to give the software users a better understanding of all the interconnected components running in the software they're deploying in their information systems. This would ultimately provide a tool to help answer the vexing "are we affected?" and "where are we affected?" questions associated with vulnerability management.

1.2 Research Motivation

While vulnerability management tools and processes have advanced considerably in recent years, there are still a number of weaknesses with the current approaches to vulnerability management.

First, as the number of vulnerabilities logged by the CVE program continues to rise exponentially, it is becoming increasingly difficult to map these vulnerabilities to an organization's affected systems particularly when the CVE records do not contain complete or timely information.

Second, with roughly only 2% of reported vulnerabilities actually exploited by malicious actors, the current approach of scanning for every single vulnerability often produces very high levels of false positives. This "noise" overwhelms human analysts and often leads to missing the more important “signals”.

Third, organizations often do not know exactly what components make up their information systems and particularly the software tools they deploy within these systems. This is the deficiency that the concept of a Software Bill of Materials (SBOM) has recently emerged to address. However, the SBOM concept does not take a systems approach to information system management, focusing entirely on software assets without taking into account hardware assets or supporting infrastructure. It also does not address the dynamic nature of a typical information system in which assets and components are often updated and moved on a regular basis.

1.3 Problem Statement

Vulnerability management systems often focus on vulnerability information for monolithic products but do not consider the product components or other metadata

associated with the systems where the products reside. And missing or incomplete vulnerability and product makeup data can lead to missed vulnerability detections.

1.4 Thesis Statement

Incorporating sub-components of a software system in vulnerability scans can lead to an increase in the number of detected relevant vulnerabilities while textual analysis of Common Vulnerability Enumeration (CVE) entry data can identify impacted software systems without relying on the associated Common Platform Enumeration (CPE) data which traditionally ties these entries to impacted systems.

1.5 Research Objectives

The primary objective of this research is to demonstrate an approach for enhancing vulnerability management systems by incorporating system sub-components and decreasing the dependency on CPE metadata. The research aims to:

- Introduce the concept of an Asset Bill of Materials (ABOM) as an extension and generalization of the Software Bill of Materials (SBOM) concept.
- Present a novel "Cards and Stacks" approach to building an ABOM.
- Propose a dynamic, asset-centric approach to vulnerability management based on the ABOM concept.
- Use CVE data along with a sample ABOM to demonstrate how our approach can lead to the discovery of more relevant vulnerability information about assets in a sample IT environment.
- Propose areas of future research on extending and enhancing the ABOM approach.

1.6 Research Questions and Hypothesis

RQ1: Can the use of metadata to identify a software asset's sub-components improve the identification of vulnerabilities relevant to a specific software asset?

RQ2: Can a text classification machine learning model be used to identify vulnerabilities relevant to a specific list of software assets based on CVE entry descriptions only absent CPE data?

H1: Augmenting a software asset with sub-component metadata will improve the identification of vulnerabilities related to the asset.

H2: A machine learning based textual analysis of CVE descriptions will be able to identify CVEs relevant to a specific asset.

1.7 Scope of Research

This research focuses on examining the vulnerabilities related to a sample enterprise IT asset stack consisting of a Lenovo laptop with an Intel microprocessor, a Microsoft Windows desktop operating system, and seven common enterprise software tools:

1. **Microsoft Word**: the most popular word processing software and part of the Microsoft Office suite of products developed by Microsoft Corp.
2. **Google Chrome**: the most popular Web browser on the Windows platform developed by Google Inc.
3. **Mozilla Firefox**: a popular alternative open-source Web browser for the Windows platform developed by the Mozilla Foundation.

4. **Adobe Acrobat Reader DC**: the most popular viewer for Portable Document Format (PDF) documents on Windows developed by Adobe Inc.
5. **VLC Media Player**: the most popular open-source media player for the Windows platform developed by the VideoLAN Project.
6. **7-Zip**: a popular open-source file compression software for the Windows platform supporting multiple file format (.zip, .7z, .rar. etc.) developed by independent software developer Igor Pavlov.
7. **Zoom Desktop Client**: the Windows desktop client version of the popular video conferencing service developed by Zoom Video Communications, Inc.

The research examines all CVE vulnerability records published in the National Vulnerability Database over the three-and-a-half-year period between 1/1/2020 and 6/30/2023.

1.8 Research Limitations

This research does not consider the versioning information of the software in the examined asset stack. In the data preparation phase, CVE record relevance is determined based only on the mention of a tool's name in the text of the description field of the CVE record without considering specific versions of the tool or trying to match its CPE details. Additionally, only a subset of CVE data (from early 2020 to mid-2023) is analyzed. Finally, limited parameter tuning was performed on the selected machine learning model in the text analysis and classification phase.

1.9 Organization of Praxis

Chapter 1 introduced the background and motivation for this research. In Chapter 2, we review the literature covering machine learning analysis of CVE data as well as the state of the art in SBOM trends and tooling. In Chapter 3, we describe our research methodology and present the coding and machine learning techniques selected for performing the research. In Chapter 4 we present and discuss the results of our research. Finally, in Chapter 5, we present our conclusions and provide recommendations for future related research topics.

Chapter 2 - Literature Review

2.1 Introduction

In this research we will examine the importance of considering the context of sub-components within a broader IT asset when managing the vulnerabilities of such asset. We will propose the use of a bill-of-materials approach to enhance the discoverability of hidden vulnerabilities and will demonstrate the use of a machine-learning text classification model to predict vulnerability relevance based on CVE descriptions. As such, we needed to examine previous works and review the current research trends in the areas of vulnerability management, asset management and software bill of materials.

2.2 Asset Management

Asset management is a critical first step in the vulnerability management lifecycle as it would be impossible for an enterprise to determine what vulnerabilities exist in its IT environment if it did not first know exactly what assets make up that environment. Yet, industry research indicates that enterprises are on average unaware of up to 40% of the devices in their IT environment (Prakash, 2023). In his paper "Asset Management, The Weakest Link in Cybersecurity Risk" the author Gyan Prakash differentiates between traditional IT asset inventory, a basic IT function with no urgency in discovery or criticality in visibility, and IT cyber asset management where accurate and detailed asset visibility is critical for securing the enterprise (Prakash, 2023). The author notes that an asset has been traditionally defined as an IP-connected device but this has changed in modern IT environments with the emergence of serverless IT microservices and other non-traditional assets. He further notes that a single application may consist of groups of these services as well as traditional assets. This, perhaps, validates our decision to

propose an asset definition broad enough to encompass complex and modular modern IT services. Prakash proposes an automated Asset Criticality Risk Rating (ACRR) approach that can identify assets that are critical to enterprise business needs. The author provides a formula for calculating an ACRR score as a function of the confidentiality, integrity and availability requirements of the asset with the score serving as a mathematical approximation of the impact to the business in case that asset is compromised. The ACRR formula is as follows:

ACRR= f(Confidentiality, Integrity, Availability)

Ci = Average weight of all the Confidentiality Impact for the asset

Ii = Average weight of all the Integrity Impact for the asset

Ai = Average weight of all the Availability Impact for the asset

ISS = Impact Sub-Score

ISS = (1 -((1-Ci)*(1-Ii)*(1-Ai)))

ACRR = roundup (min (ISS * 8, 10))

The min() function returns the item with the lowest value of the items

The roundup to zero decimal

Mathematical Ranges

Ci = [0,1] ,

Ii = [0,1] ,

Ai = [0,1]

ACRR = [0 , 10.0]

Once a score is calculated it's then mapped to a qualitative rating according to the following scale:

Rating	**ACRR Score**
None	0.0
Low	0.1 to 3.9
Medium	4.0 to 6.9
High	7.0 to 8.9
Critical	9.0 to 10.0

The author finally asserts that combining these scores with other asset attribute data can ultimately lead to better cybersecurity risk management but does not offer concrete examples or evidence of this assertion.

Matsumoto et al. also examined the importance of asset management in cybersecurity defense in their paper "Asset Management Method of Industrial IoT Systems for Cyber-Security Countermeasures" (Matsumoto et al., 2021). However, they focused their research on industrial internet-of-things (IIOT) systems. They deemed these IIOT systems harder to manage and track using traditional IT asset management systems due to their myriad network protocols, device configurations and open technologies. The authors proposed a method for streamlining the configuration and management of such

assets to support the automation of defensive cybersecurity measures. They took into consideration the following unique requirements of IIOT devices:

- **Real-time performance**: asset management methods must not disrupt the real-time communication and control processes of these devices.
- **Frugal resource usage**: the devices often have very limited power and compute capabilities so any management solution must take that into account and not burden the devices.
- **High reliability and high availability**: these devices are meant to operate reliably for extended periods of time without any downtime. Therefore, any security updates or management control software cannot require the devices to stop or restart as is common with traditional IT assets.

The authors used SysML to model the configuration of a general control system and a corresponding asset configuration management agent. They also used a CPE-based approach to define the data structures used to collect and process the asset configuration data. The figure below shows an example of an IIOT system structure and its corresponding XML representation:

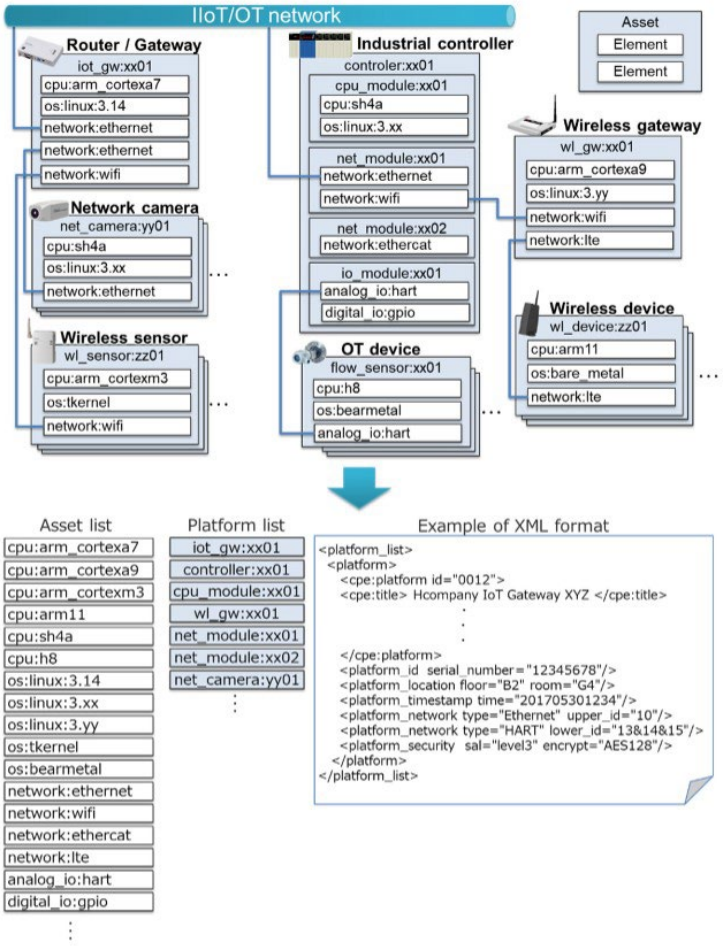


Figure 5. Example of IIOT System Structure and XML representation

The authors noted that, due to the unique resource constraints of IIOT systems discussed earlier, they had to limit their asset information collection and settled on the following configuration parameters which they deemed of high priority for effective security measures:

- Individual identification information such as manufacturer's information, serial number, etc.
- Network connection information.

- Software configuration information
- Information on security settings

Finally, the authors built a prototype of the proposed asset configuration management agent, evaluated its performance on some sample IOT devices and presented their findings.

In May 2021, the UK National Cyber Security Center (NCSC) published updated asset management guidance in recognition of the critical role of asset management in cyber security defense (UK National Cyber Security Centre, 2021). In an associated blog post announcing the updated guidance, the NCSC emphasized the importance of asset management and gave examples of situations where lack of asset management can lead to serious incidents (D., 2021). One such example describes an IT server sitting in an enterprise data centers that no one wants to touch because "nobody knows what it does" when, in fact, the server contains sensitive credentials that could be exploited by an attacker. Another example illustrates the challenges of asset management due to "shadow IT" practices, where some parts of an organization, in need of more agility than the IT department can provide, resort to purchasing and managing their own IT assets without having the needed tools or expertise to properly secure and manage such assets. Echoing some of the findings of Matsumoto et al., the NCSC acknowledges the particular difficulty of managing unusual types of Operational Technology (OT) assets within "a variety of different systems, each designed, built and operated by different teams and incorporating whatever technology was flavour of the month at the time" or managing "an environment consisting of hundreds of different types of devices, each providing its own subset of information in a non-standard format" (D., 2021).

To outline a good asset management approach, the updated asset management guidance (UK National Cyber Security Centre, 2021) provided a set of cybersecurity considerations that should be taken into account when designing an asset management system:

- **Asset discovery**: automated tools should be used to scan the environment when possible.
- **Authoritative information**: maintain an authoritative record of the environment that everyone agrees upon.
- **Accurate information**: collect and update asset information regularly so the data is never stale.
- **Availability of information:** ensure collected asset information is made available to users and systems that need it possibly through the use of a Configuration Management Database (CMDB)**.**
- **Human factors**: accommodate the needs of users such usability and accessibility.
- **Automation**: use automation to update records wherever possible.
- **Completeness**: ensure all assets are accounted for.
- **Comprehensive visibility**: ensure all critical and required asset details are collected (for example, detailed software version information).
- **Change detection**: ensure changes in the environment are detected and asset details are updated accordingly.
- **Confidentiality**: consider the sensitive nature of the asset data collected
- **Asset classification**: consider defining and using categories to streamline the grouping and management of similar and related assets.

After identifying the important assets in an IT environment with the aid of a proper asset management system, the vulnerability management lifecycle then proceeds with attempts to identify the potential vulnerabilities within those assets.

2.3 Vulnerability Management and CVE Issues

Previous works have examined the vulnerability management lifecycle with a specific focus on the Common Vulnerability and Exposures (CVE) records and the process leading to their inclusion in the NIST National Vulnerability Database. Researchers have concluded that the process of identifying and publishing vulnerabilities through CVE records and the NVD has evolved over time but remains far from perfect.

In their research paper "Cleaning the NVD: Comprehensive Quality Assessment, Improvements and Analyses" Anwar et al. identified a time lag between when the CVE was first made public and the publication date of the CVE in the NVD (Anwar, Abusnaina, Chen, Li, & Mohaisen, 2022). The researchers found that 28% of all vulnerabilities have a time lag of over a week and 70% have a time lag of six days or less, while only 38% had no such time lag. The existence of such time lags hints at the possible benefits of analyzing the CVE descriptions immediately as they are made public as opposed to waiting for the expert analysis and labeling done prior to publication in the NVD. The researchers also investigated inconsistencies in product and vendor naming conventions, resulting in over 3000 products with inconsistent names. For example, CVEs related to vulnerabilities in the Internet Explorer web browser may refer to it as "internet explorer", "internet-explorer" or "internet_explorer". These findings highlight the importance of the CPE bindings but also of contextual awareness of such issues within the data when using it in automated vulnerability scanning tools. Anwar et al.

also noted product version inconsistencies in the NVD as demonstrated by Nguyen and Massacci and briefly referenced how Dong et al. researched the use of NLP methods to find and fix these version inconsistencies using the NVD reference URLs data fields.

Similar to the work by Anwar et al., Sanguino & Uetz identified four primary issues affecting vulnerability management systems that rely on CPE information in CVE feeds:

1. **CVE entries without corresponding CPE details**: At the time of their research the authors identified 895 CVE entries without corresponding CPE details. This is a problem because vulnerability management systems that rely on CPE identifiers would not be able to find related CVE records and thus deliver incomplete results. The authors cited CVE-2016-9748 as an example. This is a vulnerability affecting versions 5.0 and 6.0 of the IBM Rational DOORS Next Generation product. While the CPE dictionary had entries for both versions, neither were matched to the CVE record at the time of the research which could lead an automated vulnerability management system to miss this CVE.
2. **Software products without assigned CPE entries**: The authors also discovered products with associated vulnerabilities and CVE records but no corresponding CPE entries in the CPE dictionary at the time of their research. They cited the example of the Microsoft Search Server product which was impacted by the vulnerability CVE-2008-4032 but did not have a CPE dictionary entry at that time. This would impact the performance of any automated vulnerability management system that tries to match the CVE to an official CPE entry leading to incorrect CVE detections (false positives) or missing detections (false negatives).

3. **CPE dictionary deprecation process**: The authors noted that version 2.3 of the CPE specification identifies three reasons for deprecating CPE entries: 1. Identifier name correction, 2. Identifier name removal and 3. Additional information discovery. At the time of their research, the authors noted the CPE dictionary contained 2,614 deprecated entries, all due to name corrections. The authors cited the example of a corrected entry for the Adobe Flash Player for the Linux operating system. This presents a problem because CVEs containing the correct identifier may not be properly matched if a vulnerability management system had already assigned a CPE identifier to a software product and that identifier is later corrected.

```
cpe:/a:adobe:flash_playe_for_linux:9.0.115.0 (deprecated)
cpe:/a:adobe:flash_player_for_linux:9.0.115.0 (corrected)
```

Figure 6. Example of deprecated CPE entry due to an identifier name correction

4. **NVD Synchronization**: The authors found that there are often discrepancies between the CPE entries in the official CPE dictionary and their supposed matches in the CVE feeds. In their analysis, they found 105,591 CPE entries in the CVE feeds that did not exist in the official CPE dictionary as of the time of their research. The authors noted that this is often the result of the different approaches to product naming taken by the CVE publishers and the official CPE dictionary maintainers. As an example, the authors noted the case of a product (Digium Asterisk) whose beta version was identified as 1.4.0_beta1 in CPE in the CVE feeds and as 1.4.0:beta1 in the CPE in the official CPE dictionary. This

indicates that CPE records are not always synchronized properly between the official CPE dictionary and the CVE records although both are hosted in the NVD maintained by NIST.

```
cpe:/a:digium:asterisk:1.4.0:beta1
cpe:/a:digium:asterisk:1.4.0_beta1
```

Figure 7. Semantically equivalent CPEs with different WFN attributes between the official CPE dictionary (first entry) and the CVE records (second entry)

Elbaz et al. coined the term "one-day vulnerabilities" (in a nod to the well-known term zero-day vulnerabilities) to refer to newly disclosed vulnerabilities that are still in the early critical phase of their lifecycles (i.e. prior to system administrators having a chance to patch or mitigate them) (Elbaz, Rilling, & Morin, 2019). The researchers proposed an approach to identify impacted software products by performing a text analysis of the human-readable text descriptions of CVEs using Term Frequency-Inverse Document Frequency (TF-IDF) techniques. Though similar to our proposed approach for attempting to predict CVE relevance using analysis of CVE descriptions, their approach does not target a specific list of products (for example, software tools) or take into consideration the context of those products within a broader enterprise IT asset. The method suggested by the researchers had a relatively low accuracy rate of 65.95% in successfully identifying one of the top three CPE associated with an analyzed CVE record.

Wåreus et al. proposed using the machine learning technique of Named Entity Recognition (NER) to automate the process of matching CVEs to their corresponding

CPEs using the CVE text descriptions (Wåreus & Hell, 2020). The authors used a Bidirectional Long Short-term Memory (BLSTM) neural network model to create the CPE labels. The CVE descriptions were first converted into vector representations to create embeddings that would allow the system to semantically understand the words. These word embeddings were then fed into the neural network to create the CPE labels. The authors model had a precision of 0.857 in achieving this task compared to a 59% success rate when attempting to achieve the same goal using simple regular expression-based methods.

Glanz et al. found that missing or inaccurate version information diminished the value of CVE records by making it more difficult to accurately determine which software systems are impacted by a vulnerability (Glanz, Schmidt, Wollny, & Hermann, 2015). Similar to the work by Wåreus et al. they proposed a solution that would automatically create the corresponding CPE for the CVE records with the missing information. However, the authors in this case used a rule-based approach to extract the relevant information from the text descriptions. Testing their approach with 300 randomly chosen CVE entries, the authors' proposed algorithm had an F1-score of 0.70 (precision of 0.71 and recall of 0.69). These results are demonstrably worse than the machine language approach used by Wåreus et al. The authors attributed the most common errors they found to non-standardized usage of CPE identifiers specifically around product sub-version information such as beta or early release numberings.

2.4 Software Bill of Materials

The concept of the Bill of Materials (BOM) can be traced back to W. Edwards Deming who is often credited with inspiring the Japanese post-World War II economic boom and the rise of manufacturing approaches like Toyota's Supply Chain Management system (Carmody et al., 2021). He used the BOM to track all the parts used in creating a product so, if defects are found in a specific part, manufacturers can easily locate all affected products. The BOM approach also allowed manufacturers to better understand their supply chains and track the quality of their suppliers (Carmody et al., 2021). A Software Bill of Materials (SBOM) is similar to a product BOM and is sometimes compared to the list of ingredients found on food packaging. The ingredients list provides needed transparency about the components of a food products so that individuals with medical conditions (certain food allergies for example) or taste preferences can make more informed buying and use decisions. Similarly, a SBOM lists all the components (smaller pieces of software code often made by third-parties) that go into a modern software product (Carmody et al., 2021). The United States Department of Commerce (The United States Department of Commerce, 2021) officially defines an SBOM as follows:

"An SBOM is a formal record containing the details and supply chain relationships of various components used in building software."

Carmody et al. found that the BOM concept has been applied to software products since the late 1990s. They cited the financial services industry as a sector where SBOM were effectively applied since 2015 when a series of software supply-chain vulnerabilities forced financial institutions to re-evaluate third party software risk to their internal infrastructure. As a result, the existence of an SBOM from a software vendor

became a litmus test of the vendor's maturity, inspiring confidence in their software offerings. The authors also cited the use of SBOMs in the Energy sector since 2014. In contrast, the healthcare industry, despite its emphasis on patient confidentiality, has been slow to adopt the concept, resulting in a dearth of published peer-reviewed literature on the topic (Carmody et al., 2021).

Wallen et. al. noted that SBOMs came to the forefront of efforts to strengthen cybersecurity defenses when the President of the United States issued Executive Order (EO) 14028, *Improving the Nation's Cybersecurity* on May 12, 2021 (Wallen, Alberts, Bandor, & Woody, 2023). The EO required United States federal government agencies to enhance software supply chain security and integrity, asserting that the use of SBOMs provides transparency into the supply chain. EO 14028 directed the creation of standards, procedures and criteria for software vendors to use when providing SBOMs directly to customers or publishing them on a public website (Biden, 2021).

As a result of EO 14028, the National Telecommunications and Information Administration (NTIA) within the United States Department of Commerce published the following guidance document: The Minimum Elements of a Software Bill of Materials (SBOM). This document defines an essential list of elements to include in an SBOM, offers use cases and provides guidance on the future direction of SBOMs. The document is clear that such guidance is foundational and that more work needs to be done to expand the use of SBOMs and, more generally, enhance the concept of transparency in the software supply chain to support software security (The United States Department of Commerce, 2021).

Wallen et al. surveyed SBOM-related literature and found that the need for providing guidance on the content and format of SBOMs emerged as a prevalent theme. However, little research existed that indicated how organizations should practically plan, develop, deploy and use SBOMs. Therefore, they focused their research on what they termed the "SBOM Lifecycle" stating that increasing the utility of SBOMs can only occur after the development of best practices and supporting processes. To build such practices, the team leveraged prior work done at the Software Engineering Institute that resulted in the Acquisition Security Framework (ASF), a framework of best practices and processes for managing the security and resiliency of software applications and systems of systems.

Using ASF as a template, the team built an SBOM framework that included the following six goals (Wallen et al., 2023):

1. **Requirements**: SBOM requirements for the program are identified and managed.
2. **Planning**: a plan for developing and using SBOMs is developed.
3. **Build/Construct**: SBOM data is create for the system, subsystem and components.
4. **Deploy/Use**: Vulnerabilities are identified and managed in SBOM software components leading to reduced system risk.
5. **Manage/Support**: SBOM risks are managed for system components.
6. **Infrastructure**: SBOM practices, software and tools are selected, implemented and managed.

2.5 Summary and Conclusion

Anwar et al. (2022) and Sanguino et al. (2017) identified and documented several issues with the CVE publication process. Similar to this research, they proposed methods to address the issues including the use of NLP methods to enhance published CVE data and automatically match it to the corresponding CPE entries. However, they did not approach the problem from the perspective of protecting a specific cyber asset (i.e. an asset-centric approach). On the other hand, Prakash et. al (2023) and Matsumoto et. al (2021) explored the importance of IT asset management in cybersecurity but focused on the reduction of overall cyber risk and the configuration management of such devices in specific environments (such as IIOT environments) as opposed to broader vulnerability management. There appears to be a paucity of peer-reviewed research or published literature on asset management in the context of defensive cybersecurity. Meanwhile, the emerging research on SBOMs, such as that conducted by Carmody et al. (2021), is primarily focused on providing guidance for the content and format of SBOMs with some recent research, such as that by Wallen et. al (2023) starting to address the complete "SBOM Lifecycle".

While there seems to be a mature body of research on vulnerability management, some limited research on asset management and a recent surge in research on SBOM concepts there is practically no research exploring the combination of these three concepts to enhance cybersecurity and vulnerability management for a specific set of IT assets. We therefore note the relative novelty of our research topic.

Chapter 3 - Methodology

3.1 Introduction

In this section we begin by reviewing the Software Bill of Materials (SBOM) concept and surveying its common formats and standards. We then introduce our own Asset Bill of Materials (ABOM) concept as a broadened generalization of the SBOM concept that allows for the representation of all of the various components and sub-components that constitute a modern IT asset. We then present our "cards and stacks" model as an example reference implementation of the Asset Bill-of-Materials concept including sample JSON and CSV-based schemas for the model. To demonstrate a practical use of the "cards and stacks" model, we present a hypothetical enterprise IT "asset stack" consisting of an enterprise laptop running the Microsoft Windows desktop operating system. We include a specific set of popular Microsoft Windows-based software applications to complete our sample "asset stack". We then compare the results of performing vulnerability scanning (i.e. identifying and collecting relevant vulnerabilities, in the form of CVE records, from the NVD) on the sample asset using traditional methods with the results of using our proposed method that incorporates the details of the asset's subcomponents. We analyze the differences and clean the resulting CVE records data to create our primary datasets. We then use these datasets to test the ability of a machine learning text classifier to predict the relevance of these CVE records to our specific asset stack based only on the text descriptions of the CVEs without relying on any related CPE information. The figure below represents the steps taken to produce the results of this research.

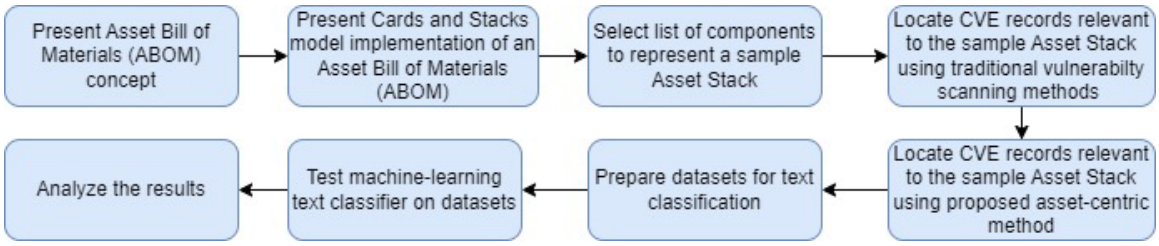


Figure 8. Flowchart of research steps

3.2 The Asset Bill of Materials (ABOM) Concept

3.2.1 Introduction

Recall that for the purpose of our research we had defined an "IT Asset" as any IT resource that provides value to the business operations of an organization including hardware, software, data, cloud resources, mobile devices, Internet-of-Things (IOT) devices and virtual computing resources. We also referenced the U.S. Department of Commerce's official definition of a Software Bill of Materials (SBOM) as "*a formal record containing the details and supply chain relationships of various components used in building software."* The Asset Bill-of-Materials (ABOM) concept is a broadened generalization of the SBOM concept that aims to provide a way to represent all the software, hardware and virtual resources constituting a modern IT asset. It aims to serve the same purpose of an SBOM by providing visibility and transparency while extending beyond software to all components and sub-components of an IT asset. The ABOM approach provides a comprehensive view of an asset to empower vulnerability management by including software (and any additional software embedded within the software), hardware, devices, systems, and platforms. We do not propose a new

standardized format specific to ABOMs but rather review the current SBOM formats and standards that can be adapted for use in an ABOM.

3.2.2 SBOM Formats and Standards

The NTIA's Survey of Existing SBOM Formats and Standards report (Tom Alrich et al., 2021) identified three key formats for SBOMs as of the report's publication date in 2021:

1. **SPDX**: the Software Package Data Exchange (SPDX) specification is an ISO/IEC standard for the exchange of SBOM information. SPDX information can be associated with a particular software product, component, set of components or even down to the level of an individual file or code snippet. The specification provides a "language" for describing an SBOM that can be represented in different file formats such as XML, XLSX, JSON and YAML. SPDX is able to capture SBOM data because it can represent all of the components found in a software product. The figure below provides an overview of an SPDX document.

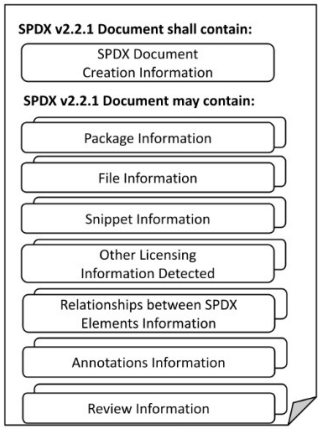


Figure 9. Components of an SPDX document

The SPDX specification also provides a rich set of relationships that can be used to link the different software elements to each other within a single document or across different SBOM documents. Each SPDX document has the following core elements:

- **Creation Information**: necessary information about the package for forward and backward tool compatibility
- **Package Information**: a package is the primary grouping of the items being described by the SPDX document which can be products, containers, components, etc.
- **File Information**: all metadata about files included in the package including their names, checksums and copyright information.
- **Snippet Information**: optional information in case specific snippets of code need to be identified separately from a parent file.
- **Other Licensing Information**: licensing or copyright information for the items being described in the document.
- **Relationships**: the SPDX format supports 43 different relationship types that can be used to describe how the packages, files and snippets within a document relate to one another or to other SPDX documents.
- **Annotations**: a place to store any additional information that does not fit within any of the other standard elements.

The most up-to-date SPDX specification information can be found at the project's website at https://spdx.dev/

2. **CycloneDX**: a lightweight SBOM specification designed primarily for software security contexts and supply chain components analysis. This project was started in 2017 with the aim to create an automatable and security-focused SBOM standard. The CycloneDX SBOMs can be represented as XML, JSON and Protocol Buffers (protobuf). The figure below provides an overview of a CycloneDX SBOM.

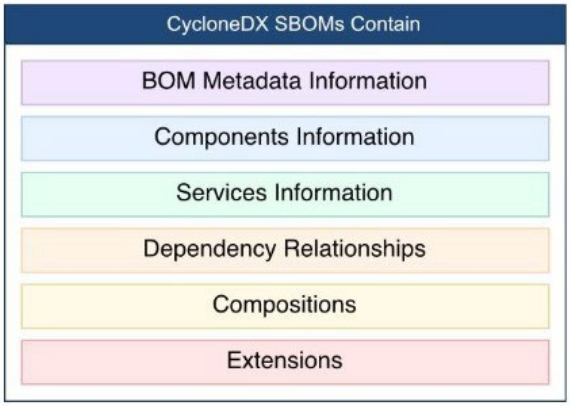


Figure 10. CycloneDX SBOM components

CycloneDX SBOMs contain the following core elements:

- **BOM Metadata**: contains metadata information such as the tool that produced the SBOM and information about the supplier or manufacturer of the software, component, firmware or device that the SBOM describes.
- **Components**: contains a complete list of the all the software components that make up the SBOM

- **Services**: describes any external APIs or Web services that the described software may rely upon including service endpoints, authentication requirements and trust boundary definitions.
- **Dependencies**: describes any dependency relationships between the components and services described in the SBOM.
- **Compositions**: a method of describing how "complete" the SBOM is (for example, incomplete first-party only, incomplete third-party only, completeness unknown etc.)
- **Extensions**: extension points allow for extending the CycloneDX specification to rapidly add new capabilities or support additional future use cases.

3. **SWID**: The Software Identification Tags (SWID) standard was defined by ISO in 2012 and updated as ISO/IEC 19770-2:2015 in 2015. SWID files commonly describe a specific release of a software product. The SWID specification defines four types of tags:
    a. **Primary tags**: to describe an installed software product)
    b. **Patch tags**: to describe an installed software patch)
    c. **Corpus tags**: to identify and describe an installable product in a pre-installed state (for example, an installer for a software product)
    d. **Supplemental tags**: an extension tag that allows additional information to be associated with any referenced SWID tag to allow third-party tools to add their own metadata.

The figure below shows a sample primary SWID tag for a fictious piece of software called “Roadrunner Detector” and published by ACME Corp.

```
<SoftwareIdentity
  xmlns="http://standards.iso.org/iso/19770/-2/2015/schema.xsd"
  xmlns:sha256="http://www.w3.org/2001/04/xmlenc#sha256"
  name="ACME Roadrunner Detector 2013 Coyote Edition SP1"
  tagId="com.acme.rrd2013-ce-sp1-v4-1-5-0"
  version="4.1.5">
  <Entity name="The ACME Corporation" regid="acme.com"
    role="tagCreator softwareCreator"/>
  <Link rel="license" href="www.gnu.org/licenses/gpl.txt"/>
  <Meta product="Roadrunner Detector" colloquialVersion="2013"
    edition="coyote" revision="sp1"/>
  <Payload>
    <File name="rrdetector.exe" size="532712"
      SHA256:hash="a314fc2dc663ae7a6b6bc6787594057396e6b3f569cd50fd5ddb
      4d1bbafd2b6a"/>
  </Payload>
</SoftwareIdentity>
```

Figure 11. Example of a SWID Primary Tag

The SWID tags can be used as an SBOM since they provide the same product or component identifying information that typically constitutes an SBOM. Tags can also link to other tags which allows them to represent dependencies as is possible in other SBOM formats.

3.3 The Cards and Stacks Model

The Cards and Stacks model presented in this research provides an example of a possible practical implementation of an ABOM. It allows us to demonstrate how the SBOM approach can be extended to asset management in practice and how the resulting ABOM approach can provide enhanced context for identifying, organizing, sorting, and prioritizing vulnerabilities. The model does not to propose a standardized ABOM format or any alternatives to current SBOM formats. In fact, it can be modified to support any of

the existing SBOM formats discussed earlier. As its name implies, our proposed model is made up of "cards" and "stacks":

- A "**card"** represents any asset sub-component that can be subject to a cyber vulnerability (for example, an open-source library utilized in a commercial software product or a particular hardware chipset component in a commercial graphics card).
- A "**stack**" consists of a group of cards that collectively represent the system we wish to analyze for vulnerabilities. Stacks may include other stacks to provide a nesting mechanism for representing more complex assets. For example, an **enterprise server asset** may be represented as a stack of software and hardware cards and stacks. Each software asset may itself be a stack of cards representing individual components such as open-source libraries or even code snippets. Each hardware asset may be a stack of cards representing individual hardware components such as memory modules, graphics cards and storage devices. A stack, therefore, consists of at least one card plus zero or more stacks.

The figure below is a visual representation of a simple device asset with one hardware component and two software components presented as cards and stacks.

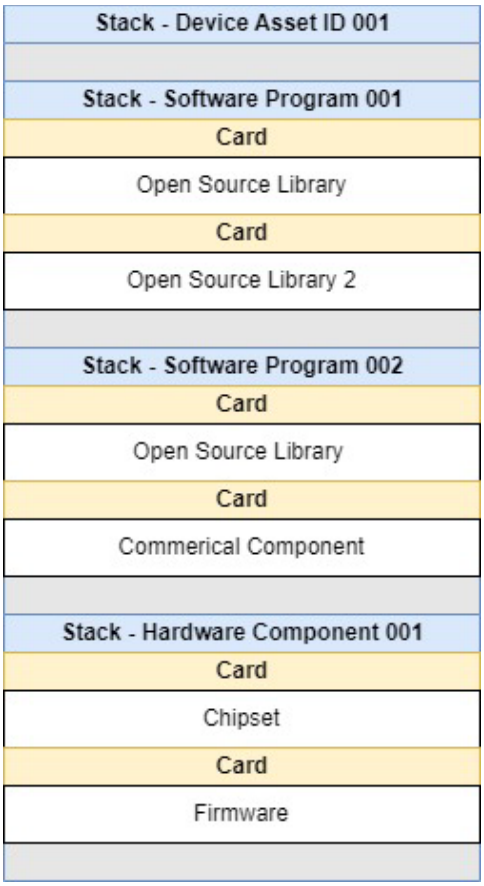


Figure 12. Visual representation of a device asset as a stack of cards and stacks

A sample asset stack representing some of the most popular software programs to run on the Microsoft Windows client operating system in an enterprise IT environment is presented in the next section and will be used as the basis for our research.

3.3.1 JavaScript Object Notation (JSON) Schema

The figure below depicts the JSON schema for the simplest scenario of a stack containing a single card:

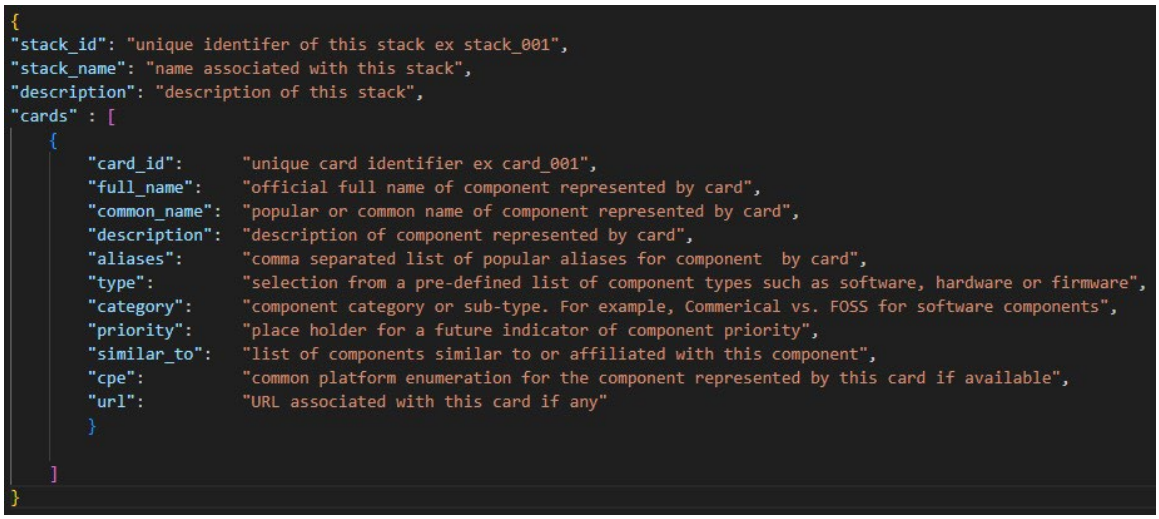


Figure 13. A stack containing a single card.

The figure below depicts the JSON schema of a more complex scenario of a stack containing one card and one nested stack with two additional cards:

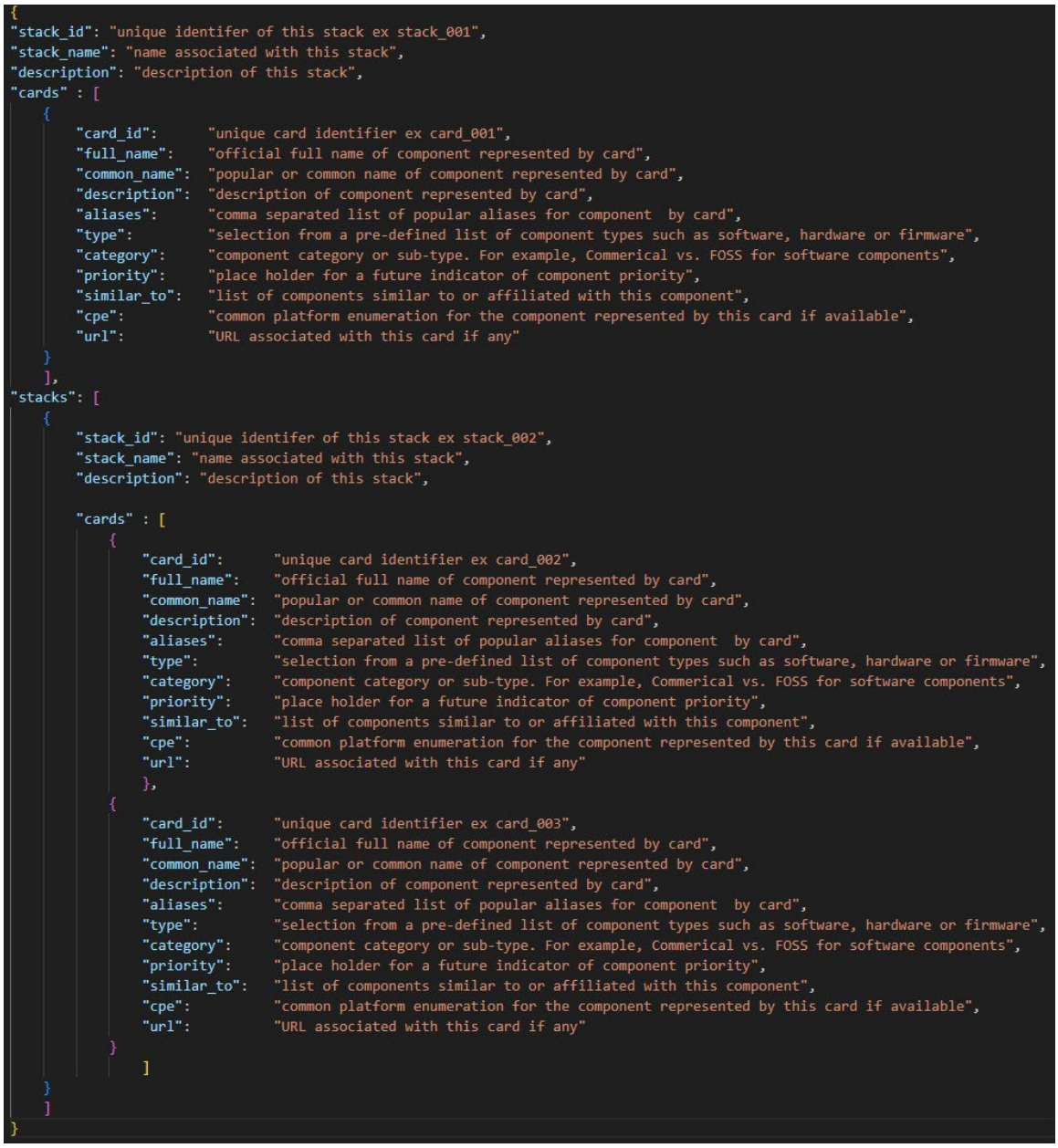


Figure 14. A stack containing one card and one nested stack with two cards.

3.3.2 CSV Schema

The most popular Python machine learning libraries often utilize data frames as the format of choice for data manipulation. Data frames are more closely aligned to the common Comma Separated Values (CSV) data format vs. JSON formats. To allow for

easier integration with our Python workflows and Jupyter notebooks, we have devised a CSV format that "flattens" the JSON hierarchy allowing for a stack to be represented in a single CSV data row. The table below describes the fields of the CSV schema that can represent the same cards and stacks data as the original JSON schema.

Table 1. Fields of the CSV format representation of the cards and stacks

Field	Description	Required
ID	ID of stack or card	Yes
Full Name	Full name of component represented by stack or card	Yes
Common Name	Popular or common name of component represented by stack or card	Yes
Aliases	Comma separated list of popular aliases for component	No
Description	Textual description of component represented by stack or card	No
Type	Component type such as hardware, software or firmware	Yes
Category	Component category or sub-type such as Commercial or FOSS	Yes
Priority	Place holder for value indicating priority of component	No
Components	List of card IDs included in the stack or blank if the row represents a card	No
Affiliates	List of components similar to or affiliated with this component	No
CPE	The NVD's common platform enumeration (CPE) for the component represented by this card if available	No
URL	URL associated with this component (if any)	No

3.3.3 Sample Asset Stack

Since our research is primarily concerned with the impact of asset sub-components on vulnerability management, we need to present a sample asset stack to test

our hypotheses and demonstrate the viability of the ABOM-based approach to vulnerability managements. We selected a group of the most popular enterprise Microsoft Windows-based software tools to represent a sample asset stack (Stephenson, 2023). Due to the popularity of the selected software, we believe this sample stack would be common in the modern enterprise IT environment (Bouchrika, 2023). The table below presents the selected software tools. For simplification, we have purposely omitted version information.

Table 2. List of software comprising the sample asset stack

Software Name	Description
Microsoft Word 365	The most popular word processing software on the Windows Platform
Google Chrome	The most popular web browser on the Microsoft Windows platform by market share
Mozilla Firefox	A popular alternative browser for Microsoft Windows PCs created and maintained by the non-profit Mozilla Foundation
Adobe Acrobat Reader DC	A popular business tool from Adobe Inc. for opening and displaying Portable Document Format (PDF) documents on Microsoft Windows PCs
VLC Media Player	A popular open-source media player for Microsoft Windows PCs
7-Zip	A popular open-source file compression and decompression tool for managing file compressions formats such as Zip and RAR.
Zoom Desktop Client for Windows	A video-conferencing tool that surged in popularity during the COVID-19 pandemic and became a common tool for remote office workers

This list of software will be the primary focus of the vulnerability data gathering and the hypotheses testing. This is the stack representing the Mozilla Firefox browser in JSON format:

```
{
"stack_id": "1002",
"stack_name": "Mozilla Firefox",
"description": "A stack representing the Firefox open-source Web
Browser by the Mozilla Foundation",
"cards" : [
    {
        "card_id":      "1002",
        "full_name":    "Mozilla Firefox",
        "common_name":  "firefox",
        "description":  "A popular open-source Web Browser by the
Mozilla Foundation",
        "aliases":      "ffox, ff, firefox esr",
        "type":         "software",
        "category":     "FOSS",
        "priority":     "high",
        "similar_to":   "edge, chrome, internet explorer, brave,
icefox, waterfox",
        "cpe":          "cpe:2.3:a:mozilla:firefox:-:*:*:*:*:*:*:*",
        "url":          "https://www.mozilla.org/en-US/firefox/new/"
    },
    {
        "card_id":      "116",
        "full_name":    "Mozilla Gecko Browser Enginer",
        "common_name":  "gecko",
        "description":  "Web browser rendering engine",
        "aliases":      "",
        "type":         "software",
        "category":     "FOSS component",
        "priority":     "high",
        "similar_to":   "blink, webkit",
        "cpe":          "",
        "url":          "https://en.wikipedia.org/wiki/Gecko_(softwar
e)"
    },
    {
        "card_id":      "117",
        "full_name":    "Mozilla SpiderMonkey",
        "common_name":  "spidermonkey",
        "description":  "Javascript and Web Assembly Browser Engine",
        "aliases":      "",
        "type":         "software",
        "category":     "FOSS component",
        "priority":     "high",
        "similar_to":   "v8,chakra",
```

```
        "cpe":          "",
        "url":          "https://spidermonkey.dev/"
    },
    {
        "card_id":      "118",
        "full_name":    "SQLite",
        "common_name":  "sqlite",
        "description":  "C-language library that implements a compact
and fast SQL database engine",
        "aliases":      "",
        "type":         "software",
        "category":     "FOSS",
        "priority":     "high",
        "similar_to":   "mssql localdb, redis db, litedb, mongo
realm",
        "cpe":          "cpe:2.3:a:sqlite:sqlite:*:*:*:*:*:*:*:*",
        "url":          "https://www.sqlite.org/index.html"
    }
    ]
}
```


Figure 15. JSON representation of the Mozilla Firefox browser stack

And this is the flattened CSV format representation of the same stack representing the Mozilla Firefox browser:

ID	Full Name	Common Name	Aliases	Description	Type	Category	Priority	Components	Similar_To	CPE	URL
116	Mozilla Gecko	Browser Engine	gecko		Web browser rendering engine	Software	FOSS component				https://en.wikipedia.org/wiki/Gecko_(software)
117	Mozilla SpiderMonkey	spidermonkey		Javascript and Web Assembly Engine	Software	FOSS component					https://spidermonkey.dev/
118	SQLite	sqlite		C-language library that implements a compact and fast SQL database engine	Software	FOSS				cpe:2.3:a:sqlite:sqlite:*:*:*:*:*:*:*:*	https://www.sqlite.org/index.html
1002	Mozilla Firefox	Firefox	ffox, ff, firefox esr	Firefox is a popular alternative browser for Windows PCs from the non-profit Mozilla Foundation							

Software	FOSS	116;117;118	Mozilla	cpe:2.3:a:mozilla:firefox:-:*:*:*:*:*:*:*
https://www.mozilla.org/en-US/firefox/new/				

Figure 16. CSV representation of the Mozilla Firebox browser stack

3.4 Python and development tools

The Python programming language is a popular choice for data science and machine learning research due to the availability of a large number of robust, free and open-source supporting frameworks and libraires. One such library is scikit-learn, a simple and efficient Python library for predictive data analytics that supports many different machine learning tasks. Scikit-learn builds upon and integrates with other popular Python libraries such as numPy, SciPy and matplotlib. We chose scikit-learn for its ease of implementation and its robust support for different machine learning models including the ones for textual classification that were needed to conduct parts of our research.

Another Python library used extensively in our research is **Pandas**. Pandas is a fast and powerful open-source library for data analysis and manipulation. Many of the data labelling and pre-processing tasks described in later sections were accomplished using Pandas.

Other prominent Python libraries used include the regular expressions (RE) management library for Python and the natural language toolkit (NLTK) for Python, which includes wrappers for powerful natural language processing (NLP) libraries and enables Python programs to interact with human language data.

Our Python code was written and organized in Jupyter Notebooks (a popular format for data science code) and executed using the Google Colab cloud-based Integrated Development Environment (IDE) and the Windows desktop-based Microsoft Visual Studio Code development environment.

3.5 Data gathering and dataset building

As described earlier, the National Vulnerability Database (NVD) is the official United States government repository of standardized vulnerability management data maintained by the National Institute of Standards and Technology (NIST) (National vulnerability database.) The primary dataset for this research consists of CVE records retrieved from the NVD.

3.5.1 NVD API

NIST provides an Application Programming Interface (API) that allows for the automated search and retrieval of CVE records from the NVD. In January 2023, NIST released version 2.0 of their API and that was the version used to retrieve the CVE records for our research dataset.

The first step in the process included requesting an API key from NVD, providing a valid email address, and agreeing to the NVD terms of service (a requirement for the use of the API). While the API can be accessed without an API key, there is a public use rate limit of five requests per 30-second rolling window. Using a validated API key increases this limit tenfold to 50 requests per 30-second rolling window.

The API allows for retrieval of CVE records using simple HTTP GET requests and returns the data in a JSON format. The HTTP GET requests can specify the values

for the different parameters supported by the API. For example, the *pubStartDate* and *pubEndDate* parameters can be used to specify a date range for when the requested CVEs records were published illustrated by the API 2.0 call below which returns CVE records that were published in the first two months of 2023:

https://services.nvd.nist.gov/rest/json/cves/2.0/?**pubStartDate**=2023-01-01T00:00:00.000-05:00&**pubEndDate**=2023-02-28T23:59:59.999-05:00

Figure 17. Sample NVD API Call

By default, each API request can return a maximum of 2,000 records. The *resultsPerPage* parameter can be used to request a fewer number of records while the *startIndex* parameter can be used to specify the first CVE record to be returned in the response. This is important since some query results may exceed 2,000 records. When this occurs, multiple subsequent API calls may be needed to retrieve all the requested records. Essentially, if the returned *totalResults* value exceeds the *resultsPerPage* parameter value (or the maximum 2,000 record limit), then additional requests using *startIndex* are needed to get the remaining records.

The sample code below illustrates the use of a looping API call to retrieve records exceeding the default 2,000 record value:

```
# Make a request to the NVD API for 2023 Q1 CVEs
url =
"https://services.nvd.nist.gov/rest/json/cves/2.0/?pubStartDate=2020-
01-01T00:00:00.000&pubEndDate=2020-04-01T00:00:00.000"
headers = {"Authorization": "Bearer " + api_key}
response = requests.get(url, headers=headers)
processed_results = 0
```

```
# Check the response status code
if response.status_code == 200:
    # The request was successful
    data = response.json()

    # Process the data here (ex. convert to CSV and save to file)

    total_results = data["totalResults"]
    results_per_page = data["resultsPerPage"]

    processed_results += results_per_page
    print ("Processed: " + str(processed_results))

    while (processed_results < total_results):

      url =
"https://services.nvd.nist.gov/rest/json/cves/2.0/?pubStartDate=2020-
01-01T00:00:00.000&pubEndDate=2020-04-01T00:00:00.000&startIndex=" +
str(processed_results)
      response = requests.get(url, headers=headers)
      if response.status_code == 200:
       # The request was successful
        print ("Processing more results...")
        data = response.json()
        results_per_page = data["resultsPerPage"]
        print ("Results Per Page: " + str(results_per_page))

        processed_results += results_per_page
        print ("Processed: " + str(processed_results))
      else:
        # The request failed
        print("Error:", response.status_code)
else:
    # The request failed
    print("Error:", response.status_code)
```

Figure 18. Looping API calls to retrive a large number of records from the NVD

3.5.2 NVDLib Python Library

The NVD API is a simple and well-documented API. However, the code to manually build and manage the HTTP requests used to call the API from Python code

can be repetitive and tedious. Furthermore, manipulating the values for date and time parameters (such as pubStartDate and pubEndDate) can be error-prone. Fortunately, there is a Python-based library (or "API wrapper") called **NVDLib** that simplifies the process of interacting with the NVD API and provides methods optimized for searching and retrieving the CVE results. For example, the following call to NVDLib's searchCVE() method will retrieve all CVEs published in the last seven days while utilizing an NVD API key:

```
import datetime
end = datetime.datetime.now()
start = end - datetime.timedelta(days=7)
results  = nvdlib.searchCVE(pubStartDate=start, pubEndDate=end, key='xxxxx-xxxxxx-xxxxxxx')
```

Figure 19. Example use of the searchCVE() API method to retrieve CVE records

Another benefit of using the NVDLib API wrapper is that it automatically handles the pagination required to work around the 2,000 record maximum records per API response. For example, the following code uses a call to the nvdlib searchCVE() method to retrieve a specified number of CVEs published within the given time interval (or ALL such records if the specified number is <= 0):

```
    if (numCVEsToGrab > 0):
        r = nvdlib.searchCVE(pubStartDate = start_date, pubEndDate = end_date, limit=numCVEsToGrab)
    else:
        r = nvdlib.searchCVE(pubStartDate = start_date, pubEndDate = end_date)
```

Figure 20. Use of custom parameters with the searchCVE() API method

3.5.3 Primary Dataset

For our primary dataset we collected CVE records published from **January 1, 2020** through **June 30, 2023**. This resulted in a dataset with a total of **78,029 records** over the given 3.5-year interval. After retrieving the JSON records from the API calls, we extracted the pertinent fields of interest and converted them to a CSV format for easier processing in our experiments. The following fields were extracted from the JSON responses:

- **ID**: the unique CVE identifier
- **Published**: the date the CVE was published
- **Description**: text description of the CVE
- **Score:** the assigned Common Vulnerability Scoring System (CVSS) score of the vulnerability (a commonly used metric approximating the relative severity of the vulnerability)
- **URL**: the URL for the vulnerability homepage on the NVD website
- **CPE**: the assigned common platform enumeration value (if any)

Below is a sample CSV record from the dataset:

ID Published Description Score URL CPE

CVE-2020-0602 2020-01-14T23:15:30.287 A denial of service vulnerability exists when ASP.NET Core improperly handles web requests, aka 'ASP.NET Core Denial of Service Vulnerability'. 7.5
https://nvd.nist.gov/vuln/detail/CVE-2020-0602
cpe:2.3:a:microsoft:asp.net_core:2.1:*:*:*:*:*:*:*

Figure 21. Sample CSV record from the primary dataset

3.6 Data labelling and clean-up

The first step of our data pre-processing was extracting CVEs relevant to the software in our sample stack using simple pattern matching techniques from the Pandas library. Our primary dataset was imported as a Pandas DataFrame which was then filtered based on matches of keywords (the software common names) with the results then exported to CSV files using built-in Pandas methods. The following code extract shows how CVEs relevant to the 7-zip software tool were extracted from the primary dataset to their own CSV file using regular expression matching techniques:

```
zip_df = df[df['Description'].str.contains('7zip|7-zip|7z',
regex=True, case=False)]
zip_df.to_csv("7zip.csv")
```

Figure 22. Code for extracting relevant CVEs

Next, we added a binary data property named "relevant" (with a default value of 0) to our dataset to indicate whether a CVE record was considered relevant to our sample stack or not. We selected a simple criterion for determining relevance: the inclusion in the text description of the CVE of the common name of the software or any of its components. In terms of our model, this meant that if a CVE description contained a refence to any of the text in the "common name" property of the cards constituting our sample stack then the CVE was deemed relevant to the stack. As mentioned before, versioning information was ignored as a matter of simplification but the process can certainly be extended to support taking the different product versions into consideration.

Pandas data manipulation techniques were then used to update the value of the "relevant" property for these records to 1. This approach resulted in identifying 2,688

relevant CVE records whose relevant property value was set to 1 and 75,341 irrelevant CVE records whose relevant property value was left as the default 0. As this dataset would be grossly imbalanced for the text analysis portion of this research, another balanced dataset was created using the "sample()" method of the Pandas library run against the entire dataset to under-sample the data and return a smaller set consisting of all the relevant records and an equivalent number of random irrelevant records. This resulted in a DataFrame with 5,376 total records evenly split (50/50) between relevant and irrelevant CVE records. That balanced dataset was then exported to its own CSV file.

3.7 Asset Stack vs. Individual Software Tool CVE Description Search

To examine our first research question of whether the inclusion of metadata to identify asset sub-components (i.e. the "asset stack" approach) will improve the identification of vulnerabilities we devised and compared two methods to search for CVE description matches:

1. The first method looked for matches in the CVE descriptions based solely on the common names of the individual software tools that make up the stack.
2. The second method looked for matches in the CVE descriptions using the common names of the software tools as well as all their components (represented by the cards making up the asset stack).

The str.contains() method of the DataFrame structure in the Pandas library was used to perform the text-based search of the CVE descriptions field using a regular expression approach and ignoring case. The following code snippet shows this approach using the individual asset names:

```
# Extract dataframe rows where the description contains one of our asset keywords
df2 = df[df['Description'].str.contains('|'.join(assets), regex=True, case=False)]
df2.describe()
```

Figure 23. Searching for CVE records using asset names only

In comparison, the following code snippet looked for matches using both the asset names as well as all their sub-components:

```
# Extract dataframe rows where the description contains one of our
asset OR component keywords

df3 =
df[df['Description'].str.contains('|'.join(assets_with_components),
regex=True, case=False)]
df3.describe()
```

Figure 24. Searching for CVE records using asset AND sub-component names

3.8 Text Analysis Approach

To examine our second research question of whether a text classification machine learning model can be used to identify vulnerabilities relevant to a specific list of software assets (i.e. our "asset stack"), we needed to determine if a classification model of CVE descriptions can predict the relevance of the CVE records to our sample asset stack. A binary classification model from the Python scikit-learn machine learning library was used to accomplish this task. To use that model, we had to first prepare our data for text analysis by transforming the text of the CVE descriptions into numerical representations that the machine learning model can use as input. To accomplish this, we utilized a series of calls to the Python regular expressions library (re) to perform the following transformations:

- Remove punctuation characters (except for periods between digits which typically signify version numbers).
- Remove all single characters.
- Substitute multiple spaces for single spaces.
- Convert to lowercase.

Once this basic data preprocessing was completed, we moved to the next step of transforming the text into a numerical representation using the Term Frequency – Inverse Document Frequency (TF-IDF) technique.

3.8.1 Term Frequency – Inverse Document Frequency

Term Frequency – Inverse Document Frequency (TF-IDF) is a simple technique for creating a numerical representation of text. It measures the importance of a term in a given document based on how frequently it occurs, or term frequency (TF), while inverse document frequency (IDF) then takes into account how common (or rare) the term is across multiple documents. The importance of the term within a document is inversely related to its frequency across documents (for example, common words such as “the” and “of” occur frequently across most documents). By calculating TF-IDF, the importance of a term in a document becomes apparent. TF-IDF is similar to the Bag of Words (BoW) technique, which counts each occurrence of a word in a document, but it is more powerful as it takes into consideration how often a word is used across all documents in the text dataset. The TF-IDF score for a term (t) can be calculated as follows:

Score = TF(t) * IDF(t)

Where:

TF(t) = (number of occurrences of the term t in a document) / (total number of terms in the document)

And

IDF(t) = log(total number of documents / number of documents that contain the term t).

In formal mathematical terms, the TF-IDF score for the word ***t*** in the document ***d*** belonging to the document set ***D*** is defined as follows (Stecanella, 2019):

$$tf\ idf\ (t, d, D) = tf\ (t, d)\ .\ idf\ (t, D)$$

Where:

$$tf\ (t, d) = log\ (1 + freq\ (t, d))$$

$$idf\ (t, D) = log\ (\ \frac{N}{count\ (d \in D : t \in d)}\ )$$

TF-IDF is useful in machine learning because it assigns a lower score to terms that occur across many documents in the dataset since such terms are likely to have lower significance (i.e. lower contribution to the meaning of the document) (Stecanella, 2019). We used the TfidfVectorizer() method of the scikit-learn Python library in combination with a list of English language stop-words from the Python natural language toolkit (NLTK) library to prepare our input data as shown in the following code snippet:

```
vectorizer = TfidfVectorizer (max_features=200, min_df=7,
max_df=0.8, stop_words=stopwords.words('english'))
processed_descriptions =
vectorizer.fit_transform(processed_descriptions).toarray()
```

Figure 25. Sample call of the TfidfVectorizer() method from the scikit-learn library

The TfidVectorizer() method was invoked with the following parameters:

- **max_features**: specifies the maximum number of features to use when building the vocabulary ordered by term frequency across the corpus. If no value is specified for this parameter, all features are used.
- **min_df**: instructs the method to ignore terms that have a document frequency strictly lower than this threshold
- **max_df**: instructs the method ignore terms that have a document frequency strictly higher than this threshold
- **stop_words**: a list of all the stop words to be removed from the resulting tokens. Here we are passing a list of English language stop words as provided by the NLTK library.

After the TfidfVectorizer() method was given the input parameters listed above, its **fit_transform()** method was used to learn the vocabulary and inverse document frequency and return a document-term matrix formatted as a Python array. The array has a shape of (5376,200) corresponding to the 5376 input rows and 200 max_features used.

3.9 Machine Learning Model description

3.9.1 Decision Tree

A decision tree is one of the most common and popular supervised machine learning algorithms. Decision trees are commonly used for classification and regression tasks. As their name implies, decision trees build a tree-like structure of "decisions" where each

internal node represents a test on an attribute, while each branch represents an outcome of a test and leaf nodes (also known as “terminal nodes”) represent a class label. The figure below shows an example of a decision tree structure.

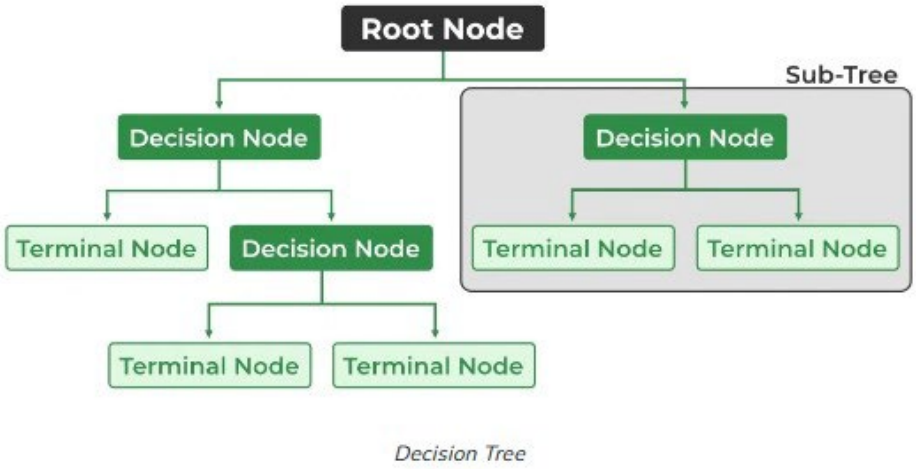


Figure 26. Decision Tree Structure

A decision tree is constructed by recursively splitting the training data into subsets based on the values of the attributes until a stopping condition (such as a maximum tree depth) is met. During training, the decision tree algorithm selects the best attribute to split the data on based on a metric such entropy or Gini impurity (a measure of the randomness in the subsets). The algorithm’s goal is to find the attributes that will maximize information gain or minimize the impurity after the split (GeeksForGeeks.com, 2023). One of the major advantages of decision trees is that they are simple to understand and interpret and require little data preparation. The trees can also be visualized. For

example, the figure below shows an example of a decision tree for determining the tastiness of a papaya fruit:

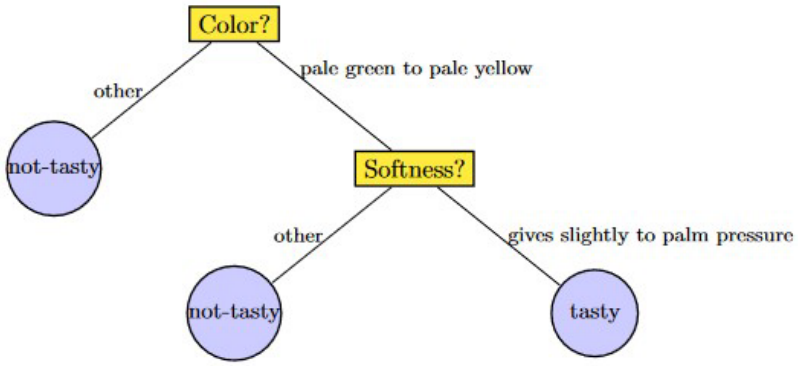


Figure 27. Decision tree to determine tastiness of a papya fruit.

One of the disadvantages of decision trees is that the decision tree learners can create over-complex trees. Another disadvantage is that the trees can be unstable because small variations in the input data might results in completely different tree structures. However, this particular disadvantage can be mitigated through the use of ensemble methods such as Random Forest (Pedregosa et al., 2011).

3.9.2 Random Forest

Ensemble machine learning methods consist of a collection of classifiers, such as decision trees, whose results are aggregated and used to select the most popular result. The most well-known examples of ensemble methods include bagging and boosting, which combine multiple models trained on different subsets of the data to reduce variance and improve predictive performance. Random Forest is an ensemble machine

learning algorithm that combines the output of multiple decision trees to reach a single result. It is technically an extension of the bagging method as it combines both bagging and feature randomness to create an uncorrelated forest of decision trees (hence the name random forest). Feature randomness is a key difference between regular decision trees and random forests. Decision trees consider all possible feature splits (i.e. decisions), while random forests only utilize a subset of those features. The figure below is a visualization of a sample random forest (IBM, 2023).

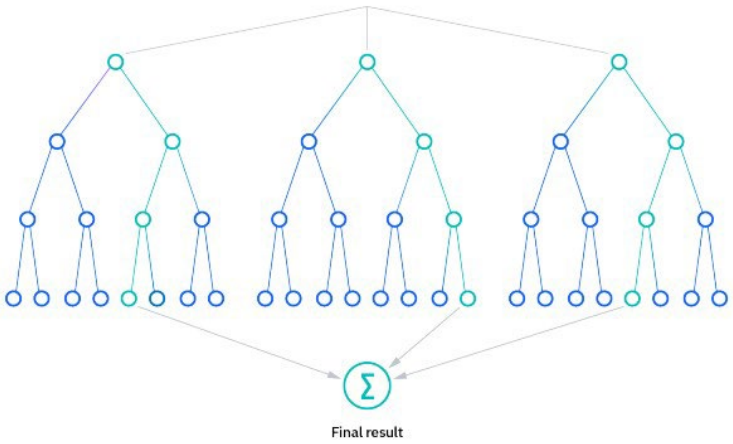


Figure 28. Sample Random Forest

Each tree in a Random Forest classifier is constructed by applying an algorithm $\boldsymbol{A}$ on a training set $\boldsymbol{S}$ and an additional random vector θ, where θ is sampled from some distribution (Shalev-Shwartz & Ben-David, 2014). Shalev-Shwartz et al. describe one method of constructing a Random Forest as follows:

"We generate θ as follows. First, we take a random subsample from S with replacements; namely, we sample a new training set S' of size m' using the uniform distribution over S. Second, we construct a sequence I1, I2, . . ., where each It is a subset of [d] of size k, which is generated by sampling uniformly at random elements from [d]. All these random variables form the vector θ. Then, the algorithm A grows a decision tree (e.g., using the ID3 algorithm) based on the sample S', where at each splitting stage of the algorithm, the algorithm is restricted to choosing a feature that maximizes Gain from the set It. Intuitively, if k is small, this restriction may prevent overfitting."

3.9.3 Applying the Model to our CVE Descriptions Dataset

After obtaining a numerical representation of CVE descriptions text data using the TF-IDF vectorizer as detailed earlier, we are able to use that data as an input to our Random Forest classifier. First, we split our labeled and processed dataset into training and test sets using a 70/30 split as follows:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(processed_descriptions, labels, test_size=0.3,
random_state=0)
```

Figure 29. Creating a 70/30 split training and testing datasets

We then use the RandomForestClassifer() method of the scikit-learn library to build our classifier and train it on our training dataset as shown in the following code snippet:

```
from sklearn.ensemble import RandomForestClassifier

text_classifier = RandomForestClassifier(n_estimators=100,
random_state=0)
```

```
text_classifier.fit(X_train, y_train)
```

Figure 30. Creating the Random Forest Classifier in scikit-learn

Finally, we generate our predictions using the test data set:

```
predictions = text_classifier.predict(X_test)
```

Figure 31. Generating the predictions using the text classifier

And evaluate the performance of our model:

```
from sklearn.metrics import classification_report,
confusion_matrix, accuracy_score

print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
print("Accuracy Score: ",  accuracy_score(y_test, predictions))
```

Figure 32. Generating a confusion matrix to evaluate the performance of our model

Chapter 4 - Results

4.1 Introduction

This chapter presents the results from the data gathering and analysis phase and hypothesis testing for our two primary research questions:

RQ1: Can the use of metadata to identify a software asset's sub-components improve the identification of vulnerabilities relevant to a specific software asset?

RQ2: Can a text classification machine learning model be used to identify vulnerabilities relevant to a specific list of software assets based on CVE entry descriptions only absent CPE data?

We present the results of searching for CVE records using the asset stack approach and the results of running the machine learning model to predict CVE record relevance using the textual description field. The code presented in our methodology was initially run in a Google Colab cloud IDE and later on a 64-bit Windows PC running Windows 11, Python 3.10 and the "Visual Studio Code" IDE.

4.2 CVE Analysis

Our primary dataset included all the CVEs published from the beginning of Q1 2020 (1/1/2020) to the end of Q2 2023 (6/30/2023) inclusive. This totaled 78,029 records divided per quarter as follows:

Table 3. Number of CVEs published per quarter.

Quarter Published	Number of CVEs
2020 Q1	4804
2020 Q2	4992
2020 Q3	4165
2020 Q4	4364
2021 Q1	4434
2021 Q2	5006
2021 Q3	5569
2021 Q4	5156
2022 Q1	6017
2022 Q2	6363
2022 Q3	6585
2022 Q4	6437
2023 Q1	7040
2023 Q2	7097

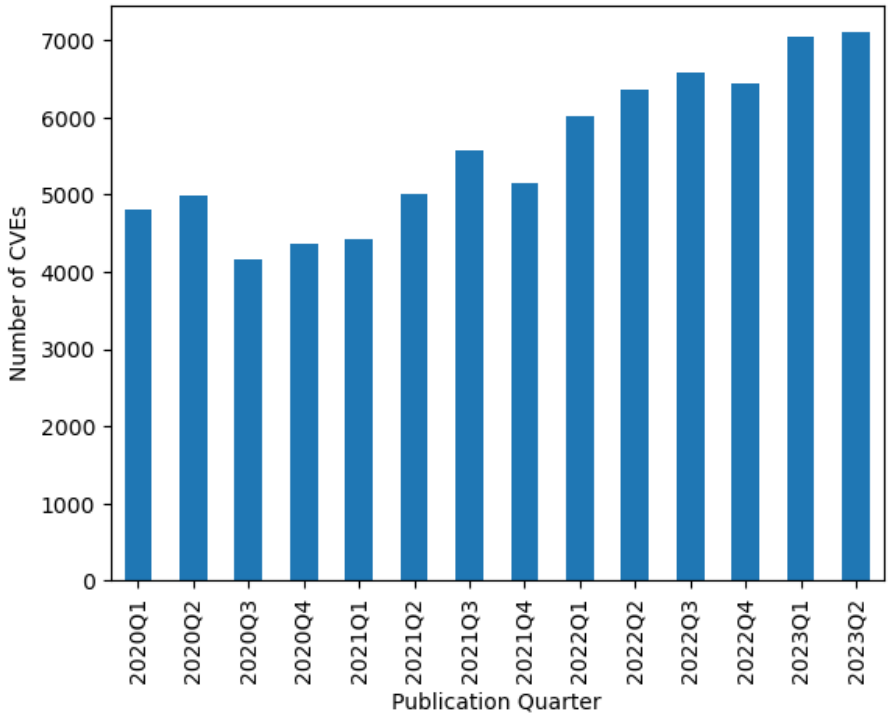


Figure 33. Number of CVEs published per quarter.

4.3 Impact of asset-centric approach on CVE discoverability

Searching the 78,029 records of our primary CVE dataset for records whose description contained a match of one or more of our seven individual asset common-names yielded 2,256 records. The searched asset names were: ‘word', 'chrome', 'firefox', 'acrobat', 'vlc', '7zip', and 'zoom’. Performing the same search while expanding the search terms to include all asset sub-components (a total of 36 search terms) resulted in 2,688 matches, an additional 432 discovered vulnerabilities for the same set of tools.

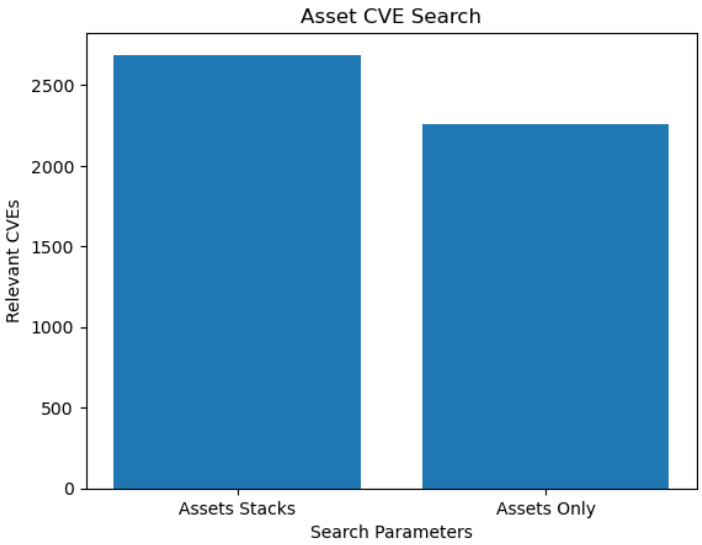


Figure 34. CVE search results for asset stacks (assets with components) vs. assets only

This 19% increase in the number of discovered vulnerabilities using the ABOM approach confirms that accounting for a software asset's sub-components can indeed improve the identification of vulnerabilities related to the asset (consistent with our H1 hypothesis).

4.4 Results of using ML text analysis for predicting CVE relevance

Our text classification machine learning model ran against our labeled and balanced dataset of 5,376 records equally split between records that are relevant and irrelevant to our asset stack. Using a 70/30 training/test split, the model attempted to predict the relevance of a CVE record to a given software tool using only the CVE description field without relying on the associated Common Platform Enumeration (CPE) value or any other parameters. Performance of our model was evaluated by producing a confusion matrix, a classification report and an overall accuracy score. The table and figure below show the resulting confusion matrix where 0 denotes irrelevant records and 1 denotes relevant records:

Table 4. Confusion Matrix of balanced dataset using Random Forest

	0	**1**
0	789	5
1	61	758

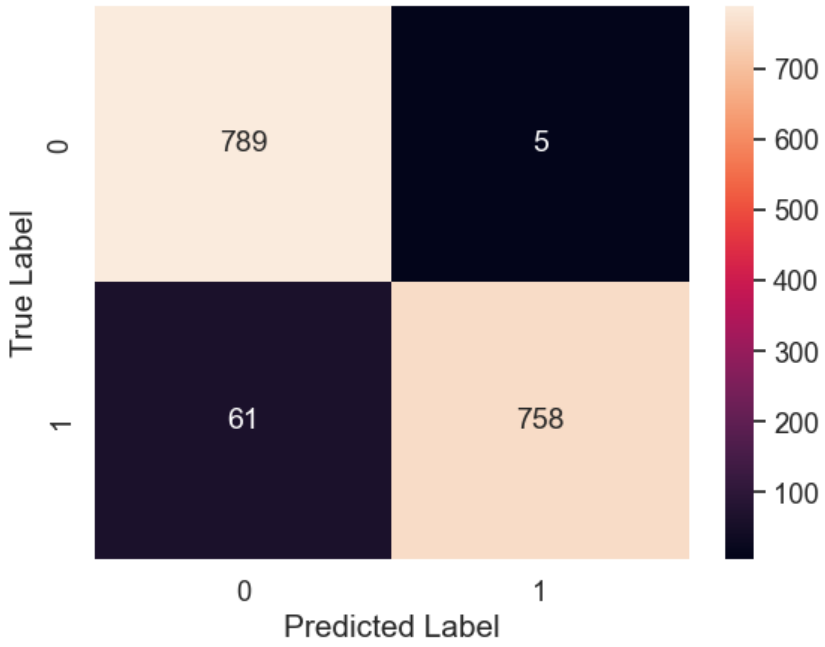


Figure 35. Visualization of confusion matrix of balanced dataset using Random Forest

The table below presents the classification report:

Table 5. Classification report for random forest model

	precision	recall	f1-score	support
0	0.93	0.99	0.96	794
1	0.99	0.93	0.96	819
macro avg	0.96	0.96	0.96	1613
weighted avg	0.96	0.96	0.96	1613

In our classification report results table:

- **0** denotes the irrelevant records
- **1** denotes the relevant records
- **Precision** is a measure of the number of correctly-identified members of a class divided by all the times the model predicted that class. In our model, , the precision score would be the number of correctly-identified relevant or irrelevant CVE records divided by the total number of predictions.
- **Recall** is a measure of the number of members of a class that our classifier identified correctly divided by the total number of members in that class. For our model, this would be the number of actual relevant or irrelevant records that the classifier correctly identified as such.
- **F1 score** combines precision and recall into one metric. A high F1 score indicates that precision and recall are both high whereas a low F1 score indicates either or both precision and recall values are low.

The resulting overall **accuracy score** for our model was **0.95908** (**~0.96)** indicating a model prediction accuracy of ~**96%**. These results demonstrate that a textual analysis machine learning model trained on CVE description data can indeed identify CVEs relevant to a specific asset with a high degree of accuracy (consistent with our H2 hypothesis).

Chapter 5 - Discussion and Conclusion

5.1 Discussion

Our results show that our proposed Asset Bill of Materials (ABOM) approach to vulnerability management provides value by enabling the discoverability of additional CVEs that may not be caught using traditional vulnerability scanning methods. Furthermore, we show that the use of a machine learning model to predict the relevance of CVE records to a specific set of software assets, based only on the text description of the CVEs, is possible.

However, there are a number of limitations to our research that we must discuss. First, our research asset stack is made up of only seven software tools which is obviously a very limited number of assets compared to what would normally be found in a corporate IT environment. This limited the number of relevant vulnerabilities that could be identified in our primary dataset to 2,688 records out of a total of 78,029 records or about 3.4% of the total dataset. This meant that we had to create a more balanced dataset for our machine learning model. We achieved this by balancing the dataset through the random selection of 2,688 irrelevant records from the original dataset. However, this balanced set is still considered very small by data science and machine learning standards and the expansion of our initial asset list to additional software and more sub-components would certainly allow for a bigger dataset. We also note that our sample stack only included software assets whereas a real-world application would likely include other asset types (such as hardware or cloud services) to realize the full potential of the versatility and extensibility of our ABOM approach.

Second, to simplify our research, we purposely ignored version numbers when dealing with the sample software assets. We noted that CVE descriptions that include affected product names also often include the version, which means our method can be extended to account for version numbers in the future and, indeed, this is a recommendation for future research.

Third, while the ABOM approach enables the use of metadata to further enhance the role of an asset's context in vulnerability management (for example, the use of the "priority" field in our sample schema to prioritize certain assets or sub-components and affect how they're handled by a vulnerability scanner), we note that our research did not study or demonstrate this capability.

Finally, since our machine learning model deals with text classification relying primarily on pattern matching, we must ask if a model based on advanced regular expressions alone (without the addition of machine learning methods) could yield similar or better results. Or better yet, could a hybrid model be constructed that uses regular expression methods to augment our machine learning approach? This too can certainly be recommended as a future research topic.

5.2 Conclusions

Vulnerability management is a critical component of cybersecurity, but the complex and dynamic nature of today's IT environments make effective vulnerability management a very challenging endeavor. Cybersecurity practitioners must have a solid understanding of the IT assets in their environment as the very first step in a vulnerability management program. The emerging software bill-of-materials (SBOM) approach provides a path for understanding and documenting the intricate dependencies between

software components. Our proposed asset bill-of-materials (ABOM) approach extends this concept to apply to entire assets --- hardware, software, virtual machines, cloud services --- to add important contextual and sub-component information to enhance vulnerability management. While vulnerability management is often considered in the context of commercial, off-the-shelf (COTS) software applications, our subcomponent-based approach enables extending vulnerability management to custom software tools that rely on open source or embeddable commercial software libraries. As an applied example, the MoveIt Transfer software vulnerability (CVE-2023-34362) highlighted earlier in our research, affected thousands of custom-developed enterprise software applications that were relying on this commercial component to perform a specific function of the software (i.e. transfer files). Our "asset stack" approach would have easily identified this library as a component of any custom application presented as a stack and allowed this vulnerability to be flagged as affecting the stack.

A related contextual challenge for effective vulnerability management is the time lag between the publication of the details of a vulnerability in an official CVE record and the inclusion of additional information regarding the affected products in the form of the CPE data added in the NVD record. For example, our primary dataset included 637 CVE records without corresponding CPE records at all. Our proposed approach would use a machine learning model that relies only on the text description of the CVE to predict relevance to our asset stack components without having to wait on the official CPE data. We showed that a model can be trained to achieve this goal with 96% accuracy using a simple machine learning method. More advanced machine learning methods (or a

combination of machine learning and regular expression-based methods as mentioned earlier) would almost certainly provide better results.

5.3 Contributions to Body of Knowledge

1. We proposed the concept of an asset bill-of-materials (ABOM) as an extension of the evolving software bill-of-materials (SBOM) concept to improve vulnerability management, which is a critical component of cybersecurity.

2. We showed how a novel "cards and stacks" approach to building an ABOM takes into consideration the context of assets (for example, asset sub-components) to allow for the discoverability of CVEs which could have otherwise been missed and presented a Python tool as a proof of concept.

3. We also showed how this approach can be combined with a machine learning method to predict relevance of CVEs to a specific collection of assets based only on the published descriptions of the CVEs.

5.4 Recommendations for Future Research

Our asset bill-of-materials (ABOM) approach was used primarily to allow for the inclusion of asset sub-components such as open-source libraries within a software application or individual chipsets within a hardware device. However, this approach is extendable to the creation of more metadata fields that would further enhance the vulnerability management process. For example, metadata fields describing an asset's status (active vs. inactive) or priority within its specific enterprise IT environment (as mentioned earlier) can be used to further refine the vulnerability analysis to raise or lower the risk levels associated with discovered vulnerabilities. As an applied example, the

infamous Log4j vulnerability (CVE-2021-44228) was frequently detected in corporate environments in both dormant software code repositories (low risk) and actively-used production software (high risk) (Log4j CVE-2021-44228 detail.2023). Future research is warranted to examine the effectiveness of additional metadata fields in an ABOM approach to help differentiate between the high risk and low risk assets, enabling cybersecurity analysts to prioritize the patching of higher-risk assets.

Asset metadata fields can also be used to map assets to the specific cybersecurity controls applied to them. Future research can examine the use of an ABOM approach to automatically assess the compliance posture of an asset to specific frameworks such as NIST 800-53 or NIST 800-171 and dynamically update the assessments as the asset status or components change. This could be a major evolution of the current (mostly static) governance, risk and compliance (GRC) assessment methods.

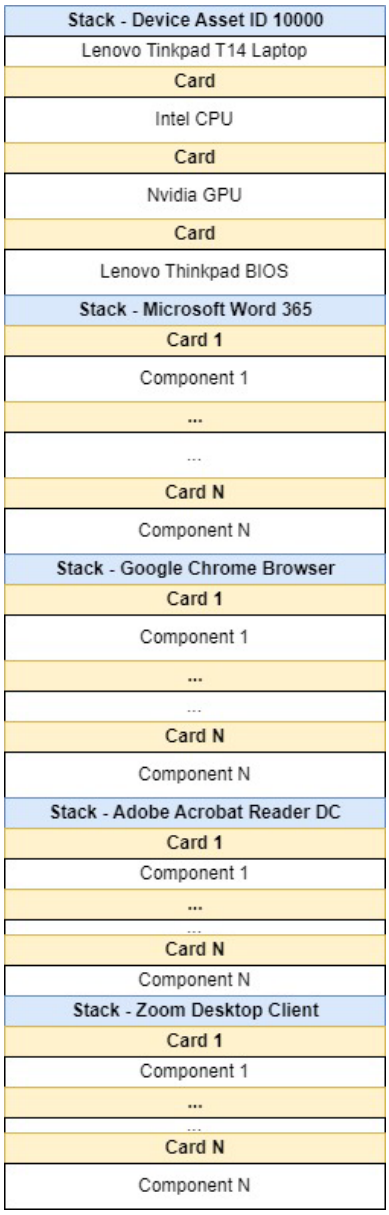
As discussed earlier, we used a basic Random Forest supervised training model for predicting the relevance of CVEs to our assets. This method achieved a decent 96% accuracy rate but future research can explore more advanced machine learning models to attempt to reach even higher accuracy rates. Additionally, since this is primarily a text-based search and analysis problem, a comparison between the use of advanced regular expressions and the machine learning approach to achieve the same results is warranted. Similarly, examination of combining the two approaches (machine learning and regular expression-based searches) could lead to additional insights.

Finally, it would be interesting to study the application of the ABOM approach to a much broader definition of an “IT asset”. The United Kingdom’s National Cyber Security Center (NCSC) includes physical locations, financial capital and even IT

personnel (and their knowledge and skills) as examples of assets that fall under the asset management umbrella (UK National Cyber Security Centre, 2021). Would it be possible then to create a single ABOM for an entire physical location (perhaps a small branch office for an enterprise) or one that includes financial as well as IT assets, for example? The dynamic and extensible nature of our proposed ABOM approach opens the door for such considerations and many others.

Appendix A – ABOM-based Vulnerability Scanner Proof of Concept

To demonstrate the potential for creating an applied vulnerability scanning tool based on our asset bill-of-materials (ABOM) approach, we created a simple proof of concept for such tool in the Python programming language. To test our proof of concept we used the following sample asset stack representing a common enterprise IT laptop:



Algorithm for the Python-based ABOM-based Vulnerability Scanner proof of concept:

1. Read a file in comma-separated values (CSV) format representing an ABOM according to the CSV specification defined in the paper.
2. Loop through each entry in the ABOM and examine its "components" field:
    a. If the components filed is blank (i.e. no components list) then this entry is a card.
        i. Query the NVD to get the CVEs related to this card using the component name as a keyword search term.
    b. Else if the components field has one or more components (cards) identified then this is a stack:
        i. Recursively loop through the components lists and query the NVD for the CVEs for each component using the component name as a keyword search term.
3. After all relevant CVEs are retrieved, loop through the results and:
    a. Output a list of all retrieved CVEs in CSV format.
    b. Output another list organized by stacks in JSON format to demonstrate the ability to nest cards within stacks.
    c. Display the total number of cards, stacks and vulnerabilities processed to the console.

Below are the sample input stack (CSV input file displayed as an Excel table), proof of concept source code and sample console output:

ID	Full Name	Common Name	Description	Type	Category	Priority	Components	Similar_To	CPE
10	Intel Core i7 Processor	intel processor	Intel Core i7 mobile processor commonly found in enteprise lapt	Hardware	CPU				cpe:2.3:h:intel:core_i7-1165g7:-:*:*:*:*:*:*:*
20	Nvidia GPU	nvidia gpu	Nvidia mobile graphics processing unit (GPU) (aka graphics card)	Hardware	GPU				
30	Lenovo ThinkPad BIOS	lenovo thinkpad BIOS	BIOS firmware used on Lenovo ThinkPad laptops	Software	Firmware				cpe:2.3:o:lenovo:thinkbook_15p_imp_firmware:-:*:*:*:*:*:*:*
100	Microsoft TypeScript Library	tslib		Software	FOSS component				
101	Microsoft ASPNet CORS	aspnet_cors		Software	FOSS component				
102	Microsoft ASPNet MVC	aspnet_mvc		Software	FOSS component				
103	Microsoft SignalR	signalr		Software	FOSS component				
104	AjaxMin	ajax_min		Software	FOSS component				
105	CSV Helper	csvhelper		Software	FOSS component				
106	Jquery	jquery		Software	FOSS component				
107	Selenium Support	selenium		Software	FOSS component				
108	UA Parser	uaparser.js		Software	FOSS component				
109	Boost	boost		Software	FOSS component				
110	SpookyHash	spookyhash		Software	FOSS component				
111	C++ REST SDK	cpprestsdk		Software	FOSS component				
112	Webkit Browser Engine	webkit	Web browser rendering engine	Software	FOSS component				
113	Blink Rendering Engine	blink	Web browser rendering engine	Software	FOSS component				
114	Google V8 Scripting Engine	v8	Javascript and Web Assembly Engine	Software	FOSS component				
115	GNOME libxml2	libxml2	XML Parser toolkit	Software	FOSS component				
119	Poppler	poppler	Poppler is a PDF rendering library based on the xpdf-3.0 code bas	Software	FOSS component				
120	Xpdf Reader	xpdf	Xpdf is a free PDF viewer and toolkit	Software	FOSS component				
121	zlib	zlib	A free open source compression library	Software	FOSS component				
128	ANTLR	antlr	A parser generator for reading, processing, executing, or translati	Software	FOSS component				
129	Apache XML Commons	xml-commons	Common code and guidelines for xml projects	Software	FOSS component				
130	libssh	libssh	A multiplatform C library implementing the SSHv2 protocol	Software	FOSS component				
131	libtext-template-perl	libtext-template-perl	A library for generating form letters, building HTML pages, or fil	Software	FOSS component				
132	libjpeg	libjpeg	A free library for JPEG image compression	Software	FOSS component				
133	sqlcipher	sqlcipher	A standalone fork of SQLite that adds 256 bit AES encryption	Software	FOSS component				
134	webidl-conversions	webidl-conversions	A javascript library to convert JS values to given Web IDL types	Software	FOSS component				
1100	Microsoft Word 365	Microsoft Word	Microsoft Word is the most popular word processing software on	Software	Commercial		100;101;102;103;104;105;106;107;108;109;110;111	Microsoft Word 2021, Word 2021, Microsoft Office Suite, Office, MS Office	
1101	Google Chrome	Chrome	Google Chrome is the most popular web browser on the Microsoft	Software	Commercial		112;113;114;115	Chromium, Chromeframe	cpe:2.3:a:google:chrome:-:*:*:*:*:*:*:*
1102	Adobe Acrobat Reader DC	Acrobat	Popular Windows tool for opening and displaying PDF document	Software	Commercial		119;120;121;115	Adobe Acrobat Pro, Adobe Acrobat DC, Adobe Acrobat Document Cloud	cpe:2.3:a:adobe:acrobat:-:*:*:*:*:*:*:*
1103	Zoom Desktop Client for Windows	Zoom	A popular video-conferencing tool that became very widespread d	Software	Commercial		128;129;130;131;132;133;134		cpe:2.3:a:zoom:zoom:-:*:*:*:*:*:*:*
10000	Lenovo ThinkPad T14 Laptop	Lenovo Thinkpad T14	A Lenovo Thinkpad T14 business laptop	Hardware	Mobile Device		10;20;30;1100;1101;1102;1103		

Figure 36. Sample stack input file for Python proof of concept

```
import pandas as pd
import nvdlib
import time
import json



# API Key for using the NVD API
api_key = "enter_api_key_here"
```

```
# Define the initial cards and stacks dictionaries to hold the
vulnerability data
cards    = {}
stacks   = {}
all_stack_vulns = {}

# Create empty DataFrame to hold our outputs
vulns_df = pd.DataFrame(columns=['Component ID', 'Component Name',
'CVE', 'CVE Description', 'CVE URL'])

count_vulns = 0


def get_cves_by_keyword(keyword) :

    # Use the NVDLib library's searchCVE() method to retireve CVEs
matching the given keyword
    r = nvdlib.searchCVE(keywordSearch=keyword, key=api_key)
    return r


def get_card_vulns(card_id) :

    # define dictionaries and lists to hold the list of
vulnerabilities associated with this card (component)
    card_vulns = {}
    vuln = {}
    vuln_list = []

    component_name = df.query(f'ID == {card_id}').iloc[0,1]
    r = cards[card_id]
    for eachCVE in r:
        cve_id = eachCVE.id
        cve_description = str(eachCVE.descriptions[0].value)
        cve_url = eachCVE.url

        # Build both the DataFrame of all vulnerabilities (for the
CSV export)...
        vulns_df.loc[len(vulns_df)] = [card_id, component_name,
cve_id, cve_description, cve_url]

        # And the list of vulnerabilities (for the JSON export)
        vuln["CVE_id"] = cve_id
        vuln["CVE_description"] = cve_description
        vuln["CVE_url"] = cve_url
```

```
        vuln_list.append(vuln)


    if (len(vuln_list) > 0):
        card_vulns[component_name] = vuln_list

    return card_vulns


def get_stack_vulns(stack_id) :

    stack_vulns = {}
    cards_vulns = []


    card_vulns  = get_card_vulns(stack_id) # get vulnerabilities of
the stack's primary card
    if (len(card_vulns) > 0):
        cards_vulns.append(card_vulns)

    # Get the list of cards (sub-components) that make up this stack
    cards_list = [int(card_id) for card_id in
stacks[stack_id].split(';')]
    for card_id in cards_list:
        if card_id in stacks: # it's a stack
            get_stack_vulns(card_id) # recursively get vulns for the
stack
        else: # it's a single card
            card_vulns  = get_card_vulns(card_id)
            if (len(card_vulns) > 0):
                cards_vulns.append(card_vulns) # get vulnerabilities
for the single card


    # Add all the cards vulnerabilities to the aggregate list of
stack vulnerabilities (for the JSON export)
    stack_vulns[stack_id] = cards_vulns
    stack_name = df.query(f'ID == {stack_id}').iloc[0,1]
    stack_vulns['stack name'] = stack_name

    return stack_vulns
```

```
print
("#################################################################")
print
("###                                                           ###")
print ("### Welcome to AVS - an ABOM-based Vulnerability Scanner!
###")
print
("###                                                           ###")
print
("#################################################################")

# input("Please enter the name of the ABOM file to process: ")


# Specify the path to our ABOM input file in CSV format
file_path = 'Proof_of_Concept_Sample_Stack2.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(file_path)

# print(tabulate(df, headers='keys', tablefmt='psql'))


for index, row in df.iterrows():

    card_id = int(row['ID'])
    cards[card_id] = get_cves_by_keyword(str(row["Common Name"]))
    count_vulns = count_vulns + len(cards[card_id])

    # if the components (cards) list for this entry is not empty then
it's a stack
    if pd.notna(row['Components']):
        # stacks.append(component_id)
        stacks[card_id] = row['Components']


print(f"Processing {len(stacks)} Stacks...")

# The primary call to get_stack_vulns() will build both the main list
of stack vulnerabilities for the JSON export
# and the DataFrame of all vulnerabilities for the CSV export
for stack_id in stacks.keys():
    stack_name = df.query(f'ID == {stack_id}').iloc[0,1]
    print(f"Processing Stack ID:{stack_id} - {stack_name}...")
    all_stack_vulns[stack_id] = get_stack_vulns(stack_id)
```

```
# Export the DataFrame of all card and stacks vulnerabilities to CSV
(will contain duplicates if a several stacks share common cards)
vulns_df.to_csv("cards_and_stacks_vulns.csv")

# Export the list of vulnerabilities (by stack) in JSON format
with open('stacks_vulns.json', 'w') as output_file:
    json.dump(all_stack_vulns, output_file)

# display brief analysis / metrics of the records processed
vulns_df.count()
count_cards = len(cards)
count_stacks = len(stacks)
print(f'Retrieved {count_vulns} vulnerabilities for {count_stacks}
Stacks and {count_cards} Cards.')
```

Figure 37. Python source code for proof of concept

```
#################################################################
###                                                           ###
### Welcome to AVS - an ABOM-based Vulnerability Scanner!     ###
###                                                           ###
#################################################################
Processing 5 Stacks...
Processing Stack ID:1100 - Microsoft Word 365...
Processing Stack ID:1101 - Google Chrome...
Processing Stack ID:1102 - Adobe Acrobat Reader DC...
Processing Stack ID:1103 - Zoom Desktop Client for Windows...
Processing Stack ID:10000 - Lenovo ThinkPad T14 Laptop ...
Retrieved 9376 vulnerabilities for 5 Stacks and 34 Cards.
```

Figure 38. Console output from a sample run of the proof of concept

References

Anwar, A., Abusnaina, A., Chen, S., Li, F., & Mohaisen, D. (2022). Cleaning the NVD: Comprehensive quality assessment, improvements, and analyses. *IEEE Transactions on Dependable and Secure Computing, 19*(6), 1. doi:10.1109/TDSC.2021.3125270

Biden, J. (2021). Executive order on improving the nation's cybersecurity. Retrieved from https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

Bouchrika, I. (2023). 15 best business software in 2023. Retrieved from https://research.com/software/best-business-software

Brotherston, L., & Berlin, A. (2017). *Defensive security handbook: Best practices for securing infrastructure* " O'Reilly Media, Inc.".

Carmody, S., Coravos, A., Fahs, G., Hatch, A., Medina, J., Woods, B., & Corman, J. (2021). Building resilient medical technology supply chains with a software bill of materials. *Npj Digital Medicine, 4*(1), 34.

Cheikes, B. A., Cheikes, B. A., Kent, K. A., & Waltermire, D. (2011). *Common platform enumeration: Naming specification version 2.3* US Department of Commerce, National Institute of Standards and Technology ….

CISA. (2021). What is cybersecurity? | CISA. Retrieved from https://www.cisa.gov/news-events/news/what-cybersecurity

CSRC Content Editor. (2023). Well-formed CPE name - glossary | CSRC. Retrieved from https://csrc.nist.gov/glossary/term/Well_Formed_CPE_Name

D., T. (2021, May 28,). Asset management for cyber security. Retrieved from https://www.ncsc.gov.uk/blog-post/asset-management-for-cyber-security

Dong, Y., Guo, W., Chen, Y., Xing, X., Zhang, Y., & Wang, G. (2019). (2019). Towards the detection of inconsistencies in public security vulnerability reports. Paper presented at the *28th USENIX Security Symposium (USENIX Security 19),* 869-885.

Elbaz, C., Rilling, L., & Morin, C. (2019). (2019). Towards automated risk analysis of" one-day" vulnerabilities. Paper presented at the *RESSI 2019-Rendez-Vous De La Recherche Et De L'Enseignement De La Sécurité Des Systèmes D'Information,* 1-3.

GeeksForGeeks.com. (2023). Decision tree. Retrieved from https://www.geeksforgeeks.org/decision-tree/

Glanz, L., Schmidt, S., Wollny, S., & Hermann, B. (2015). (2015). A vulnerability's lifetime: Enhancing version information in CVE databases. Paper presented at the *Proceedings of the 15th International Conference on Knowledge Technologies and Data-Driven Business,* 1-4.

IBM. (2023). What is random forest? Retrieved from https://www.ibm.com/topics/random-forest

International Organization for Standardization. (2022a). *ISO/IEC 27001 information security management systems.* ().International Organization for Standardization. Retrieved from https://www.iso.org/standard/27001

International Organization for Standardization. (2022b). *ISO/IEC 27002:2022 information security, cybersecurity and privacy protection — information security controls.* ().International Organization for Standardization. Retrieved from https://www.iso.org/standard/75652.html

Kelley, K. (2023). What is cybersecurity and why it is important? Retrieved from https://www.simplilearn.com/tutorials/cyber-security-tutorial/what-is-cyber-security

Kolesnikov, N. (2023). 50+ cybersecurity statistics for 2023 you need to know where, who & what is targeted. Retrieved from https://www.techopedia.com/cybersecurity-statistics

Log4j CVE-2021-44228 detail. (2023). Retrieved from https://nvd.nist.gov/vuln/detail/CVE-2021-44228

Matsumoto, N., Fujita, J., Endoh, H., Yamada, T., Sawada, K., & Kaneko, O. (2021). Asset management method of industrial IoT systems for cyber-security countermeasures. *Information (Basel), 12*(11), 460. doi:10.3390/info12110460

MITRE. (2013). CPE specifications — archive. Retrieved from https://cpe.mitre.org/specification/

MITRE. (2023a). CVE numbering authority (CNA) rules. Retrieved from https://www.cve.org/ResourcesSupport/AllResources/CNARules

MITRE. (2023b). Overview | CVE. Retrieved from https://www.cve.org/About/Overview

MITRE Corporation. (2023). About CWE - common weakness enumeration. Retrieved from https://cwe.mitre.org/about/index.html

Muniz, J., McIntyre, G., & AlFardan, N. (2015). *Security operations center: Building, operating, and maintaining your SOC* Cisco Press.

National Institute of Standards and Technology. (2023). NVD - understanding vulnerability detail pages. Retrieved from https://nvd.nist.gov/vuln/Vulnerability-Detail-Pages

National vulnerability database. Retrieved from https://nvd.nist.gov/

Nguyen, V. H., & Massacci, F. (2013). (2013). The (un) reliability of nvd vulnerable versions data: An empirical experiment on google chrome vulnerabilities. Paper presented at the *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security,* 493-498.

Page, C. (2023, August 25,). MOVEit, the biggest hack of the year, by the numbers. *Techcrunch.Com,* Retrieved from https://techcrunch.com/2023/08/25/moveit-mass-hack-by-the-numbers/

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Dubourg, V. (2011). Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research, 12*, 2825-2830.

Petrosyan, A. (2023). Annual number of malware attacks worldwide from 2015 to 2022. Retrieved from https://www.statista.com/statistics/873097/malware-attacks-per-year-worldwide/

Prakash, G. (2023, Asset management, the weakest link in cybersecurity risk. *Cyber Defense Magazine,* Retrieved from https://www.cyberdefensemagazine.com/asset-management/

RiskOptics. (2021). What is vulnerability management under ISO 27001? Retrieved from https://reciprocity.com/what-is-vulnerability-management-under-iso-27001/

Sanguino, L. A. B., & Uetz, R. (2017). Software vulnerability analysis using CPE and CVE. *arXiv Preprint arXiv:1705.05347,*

Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms* Cambridge university press.

Stecanella, B. (2019, May 10,). Understanding TF-ID: A simple introduction. Retrieved from https://monkeylearn.com/blog/what-is-tf-idf/

Stephenson, B. (2023). The top 10 internet browsers for 2023. Retrieved from https://www.lifewire.com/top-internet-browsers-4589106

The United States Department of Commerce. (2021). *The minimum elements for a software bill of materials (SBOM).* ().The National Telecommunications and Information Administration (NTIA). Retrieved from https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf

Tom Alrich, Chris Clark, Patrick Dwyer, Robin Gandhi, Adam Weinrich, Christopher Gates, . . . Steve Winslow. (2021). *Survey of existing SBOM formats and standards.* ().National Telecommunications and Information Administration (NTIA). Retrieved from https://www.ntia.gov/sites/default/files/publications/sbom_formats_survey-version-2021_0.pdf

UK National Cyber Security Centre. (2021). Asset management guidance. Retrieved from https://www.ncsc.gov.uk/guidance/asset-management

Wallen, C. M., Alberts, C., Bandor, M., & Woody, C. (2023). *Software bill of materials framework: Leveraging sboms for risk reduction.* (). Pittsburgh, PA: Carneige Mellon University. Retrieved from https://insights.sei.cmu.edu/documents/5364/Leveraging_SBOM_for_Risk_Reduction.pdf

Wåreus, E., & Hell, M. (2020). (2020). Automated CPE labeling of CVE summaries with machine learning. Paper presented at the *Detection of Intrusions and Malware, and*

Vulnerability Assessment: 17th International Conference, DIMVA 2020, Lisbon, Portugal, June 24–26, 2020, Proceedings 17, 3-22.

Wikipedia contributors. (2023). *Common vulnerabilities and exposures* Wikipedia, The Free Encyclopedia.