Sheridan
FACULTY OF APPLIED SCIENCE & TECHNOLOGY

Passive Vulnerability Management through CVE/CPE Analysis

Adrian Yip
991538262
yipad@sheridancollege.ca

Mina Abdelsayed
991444840
abdelsmi@sheridancollege.ca

Academic Advisors:

Ali Hassan
ali.hassan1@sheridancollege.ca

Syed Raza Bashir
syedraza.bashir@sheridancollege.ca

Syed Tanbeer
syed.tanbeer@sheridancollege.ca

Faculty of Applied Sciences and Technology, Sheridan College
1430 Trafalgar Rd, Oakville ON, Canada

August 16th, 2022

Abstract

As the number of softwares found to be installed on any system continues to increase, so do the number of vulnerabilities that could be taken advantage of too. In recent years, we've seen how software vulnerabilities could wreak havoc on systems and businesses, such as that of recent log4j. This has stressed the importance of awareness of publicly shared vulnerability information, especially those known to be trustworthy sources such as the National Vulnerability Database (NVD). However, there are currently little ways to take advantage of this data and sources to protect a system. The method of doing so that is being explored is through passive detection and management. This is an important distinction from traditional active scanning protection methods, as it also allows alleviating important system (or asset) resources since it does not require intense analysis and scanning methods that also take time. This is increasingly important in critical environments, especially when there are many assets to manage. Our approach is to explore passive detection and management, in the context of vulnerability information. This is done through leveraging previous work exploring string matching algorithms, to match the software inventory data and profile it to matched CPE and CVE data. We explore methods of extracting this software inventory data from systems, as well as, providing much needed tools of management for a passive vulnerability detection system.

Keywords

Passive detection, software vulnerability, management system, software inventory tracking.

Table of Contents

Glossary of Terms and Abbreviations

Glossary

Vulnerability Detection and Management	A process in the cybersecurity operations that involves detecting known vulnerabilities within software used in an environment. Detected vulnerabilities are indexed against public resources and assigned a score. Managing vulnerabilities involve tracking the status, remediation and providing updates and alerts.
Active Scanning	Active Vulnerability Scanning is the process of detecting vulnerabilities through intrusively and disruptively sending detection requests to elicit a response. Scanners will analyze the response to determine if it indicates a vulnerability exists.
Passive Detection	Passive Detection is a scan-less and non-disruptive method of detecting vulnerabilities through comparison of software versions and vendors. Vulnerable software metadata is tracked in a central database such as the NVD and compared with user software information.

List of Abbreviations

API	Application Programming Interface
AVS	Active Vulnerability Scanner
CMDB	Configuration Management Database
CPE	Common Platform Enumeration
CSV	Comma-Separated Values
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
NER	Named Entity Recognition
NIST	National Institute of Standards and Technology
NLP	Natural Language Processing
NVD	National Vulnerability Database
PVD	Passive Vulnerability Detection
WFN	Well-Formed CPE Name
VM	Virtual Machine

1. Introduction & Background

1.1. Problem Statement

Vulnerability management refers to a process in information security where vulnerabilities are identified, mapped to specific assets, and some form of remediation is performed. The increasing amount of dependency on software systems has led to malicious actors exploiting weaknesses within the programs that enable organizations to operate [1]. There are 2 main types of vulnerability identification: Active Vulnerability Scanning (AVS) and Passive Vulnerability Detection (PVD). Currently the market is heavily saturated with AVS solutions with offerings such as Nessus, OpenVAS, Qualys, etc. There are very few Passive Vulnerability Detection products available with many only seen in research and studies.

Research has shown that PVD produces more accurate results than AVS when it comes to false positives in Vulnerability detection [2]. In addition, AVS has certain drawbacks including coverage, speed, and scalability [2] which are often magnified for smaller to medium sized organizations. AVS often fails to perform tests in a timely manner, is unable to cover the full network topology due to certain configurations and can often impact the performance of the network environment. All of these drawbacks are not present or minimized in a PVD system.

The main problem currently is that there are no robust PVD systems that allow administrators to efficiently import software inventory data, analyze and track vulnerabilities. This project will aim to remediate such struggles by building a framework to ingest such data, analyze vulnerabilities from the National Institute of Standards and Technology's (NIST) National Vulnerability Database (NVD) and enable the user with a set of tools to manage the discovered vulnerabilities [3].

1.2. Project Goals and Outcomes

The goals of this project are three-fold:

1) System for Identifying Software and Versions

A system for parsing software inventory and asset data is required for further analysis and comparison against the NVD. Currently there are tools and websites which allow users to look up specific software in the Common Platform Enumeration (CPE) format to retrieve information about vulnerabilities. However, these do not provide the capability to import data that one would already have from a software repository or a Configuration Management Database (CMDB). Such a system in the project will be able to read exported data from commonly used software repositories such as Open-Source/Proprietary CMDB, Software Inventory Tools and Profilers, Network Scans, OS Scripts. The read data will then be passed to the CPE profiler and Common Vulnerabilities and Exposures (CVE) lookup system.

2) Improved CPE string matching lookup

Previous works have noted that matching descriptions of software and their version to the CPE format is not entirely accurate. Due to the manual update nature of the database, CPE identifiers are tagged to vulnerabilities on a rolling basis leading to inconsistency [2]. In addition, research has suggested that manual interaction is preferred to validate the CPE matches [4]. Building from other studies, improved string matching algorithms such as Ratcliff/Obershelp will be explored to create a CPE lookup that is efficient and accurate.

3) Interface and system for Vulnerability Management activities

Following CPE and CVE profiling, users should also be provided with a set of tools to manage such vulnerabilities in the system. A notification and alert system should be developed to notify users that an updated entry in the NVD database has been detected for specific software they have imported or marked to follow. In addition, a status monitor to track remediation processes and notes will also be prioritized. Time and project scope permitting, further features can be considered.

The main contribution of this project will be a fundamental framework and system that imports user software data to present associated vulnerabilities through the creation of a passive vulnerability management system. It will build in improvements to CPE string matching and ease of user configuration.

1.3. Prior Work/Literature Review

Published work and research on Software Vulnerability Analysis using CPE and CVE has been done a few times in the academic world. Some of this research focused on the viability of Passive Vulnerability Detection and Management while others looked at String Matching approaches to provide accurate results when searching the NVD. The following prior work and literature will be split into two categories: 1) PVD Research and 2) String Matching Analysis.

1) PVD Research

In 2021, Harun Ecik studied the capabilities and accuracies of both AVS and PVD. His research indicated that traditionally a large majority of vulnerability detection was performed using AVS. Through his analysis, he indicates that AVS has certain shortcomings. These include AVS coverage of vulnerabilities as the most popular brand Nessus, only accounts for 35% of all vulnerabilities identified in the NVD. In addition, AVS scanners are prone to visibility and scalability impediments. They are often difficult to configure properly to scan the entire network environment and can lead to unrealistically long scan times when many hosts are involved. The scanning can also lead to possible network disruptions and downtime. PVD on the other hand can account for almost all vulnerabilities in the NVD and doesn't require as many resources as AVS.

However, this is not to say the PVD is the end all be all for vulnerability detection. It also comes with its own flaws such as timeliness of detection due to reliance on the NVD publishing vulnerabilities as well as reliability of results. Continuing from this, Ecik experiments with the accuracy of results from PVD and AVS detection in a controlled lab environment. Using scans and analysis of target virtual machines (VM) with verified hardware and software, the vulnerability detection results were compared. Results showed that AVS accuracy was worse than PVD accuracy especially on Linux systems [2].

The 2017 paper by Rafael Uetz and Luis Sanguino performs the initial test of developing a vulnerability analysis system using the CPE and CVE identifiers from the NVD. They identified certain shortcomings of the CPE and CVE system which included delayed mapping, some missing relationships and inaccuracy of CPE labeling when done automatically without human interaction. Their research identifies an approach for breaking up CPE strings into various fields to be used as search terms. CPE identifiers are searched using the Levenshtein distance string matching algorithm and then matched with appropriate CVE fields.

This study displays certain issues with the current matching approach. CPE identifiers are often matched incorrectly when automated and CVE results can be misleading if the NVD provides delayed or inaccurate results [4].

2) String Matching

The initial study from 2017 as referenced by [4] used the Levenshtein distance algorithm for matching CPE identifiers. This string matching algorithm prioritized fields extracted from strings for the vendor and name of the software product. The distance algorithm is a way of determining the difference or “distance” between two strings. It identifies this by determining the number of changes needed to modify one string to another. This includes inserting characters, deleting characters or substituting. This study in particular used a Levenshtein distance of 2 or less. However, this method was often inaccurate when run fully automatically. Hence, manual interaction was preferred to avoid inaccuracies.

Another paper from Daniel Tovarňák et al. studied the use of a graph-based model for matching CPEs with vulnerabilities. The research takes two Well Formed Names and performs graph-based attribute matching to identify the relationship between the strings.The query compares both the CPE name and then searches for associated CVE values. The paper identifies the graph insertion procedure and model used for such matching. It does not identify accuracy, testing results or possible problems with the suggested method [6].

Building upon previous attempts at the problem, Roman Ushakov et al. used the Ratcliff/Obershelp algorithm. This algorithm takes two strings and calculates the similarity using metrics such as the number of matching characters and the total number of characters from both strings. From the study, the algorithm would identify CPEs with a similarity of 0.95 or higher. The end results of the system indicated that it could successfully match 79% of the CPEs of the 26 application sample size [5].

Lastly a study in 2020 researched the possibilities of labeling CPEs from the CVE summary section using machine learning. Through the use of Natural Language Processing (NLP), Named Entity Recognition (NER) was performed to identify data labels from the CPE data structure. Data labels and a dataset from the NVD were used to train the model and identify CPEs with a precision of 0.8604. This resulted in a reconstruction of CPEs for 67.44% of the CVE summaries used [7].

A summary of the prior works, their methodology, and the problems that were identified as areas for further research are presented in Table I.

TABLE I
Prior Work/Literature Comparison

Prior Work/Literature	Methodology	Problems Identified/Areas for Further Work
Ecik, H (2021) [2]	Comparison of AVS and PVD systems and their accuracy	● Additional testing for larger set of CPE
Sanguino, L et al. (2017) [4]	Development of PVD system using Levenshtein algorithm for matching	● CPE manual interaction ● CVE error validation ● Improved matching accuracy
Tovarňák, D et al. (2021) [6]	Development of vulnerability matching system using a graph-based approach	● Comparison of accuracy against other approaches
Ushakov, R et al. (2021) [5]	Development of vulnerability matching system using Ratcliff/Obershelp algorithm for matching	● Additional testing for larger set of CPE ● Improved matching accuracy
Wåreus, E et al. (2020) [7]	Development of NLP system to identify CPEs from CVE summaries	● Improved matching accuracy ● Joining with other systems for larger CPE identification coverage

2. Hypothesis/Proposed Solution(s)

This project's proposed solution will entail building a passive vulnerability management system that incorporates and builds upon previous research done on the topic. The structure of such a system can be seen as in Figures 1 to 4.

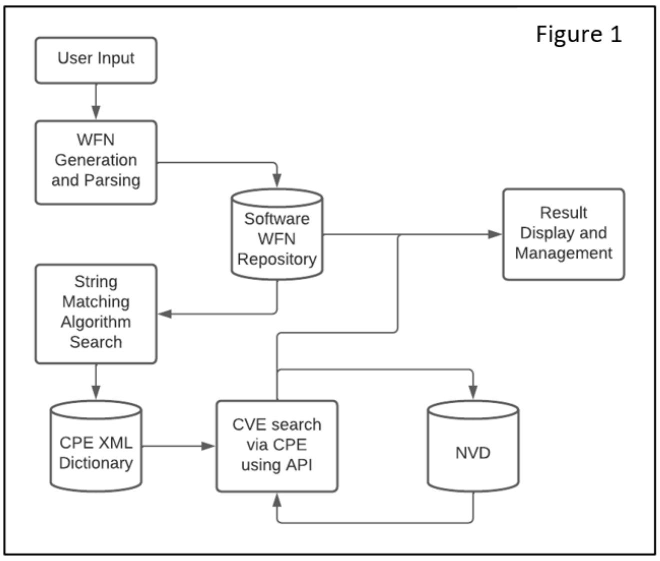
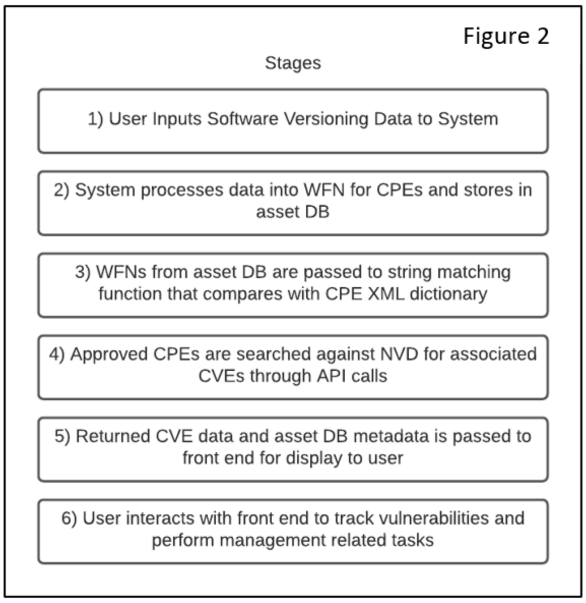


Figure 1
User Input
WFN Generation and Parsing
Software WFN Repository
Result Display and Management
String Matching Algorithm Search
CPE XML Dictionary
CVE search via CPE using API
NVD

Fig. 1. Passive Vulnerability Management System Environment



1) User Inputs Software Versioning Data to System

2) System processes data into WFN for CPEs and stores in asset DB

3) WFNs from asset DB are passed to string matching function that compares with CPE XML dictionary

4) Approved CPEs are searched against NVD for associated CVEs through API calls

5) Returned CVE data and asset DB metadata is passed to front end for display to user

6) User interacts with front end to track vulnerabilities and perform management related tasks

Fig. 2. Step by Step process for the Passive Vulnerability Management system

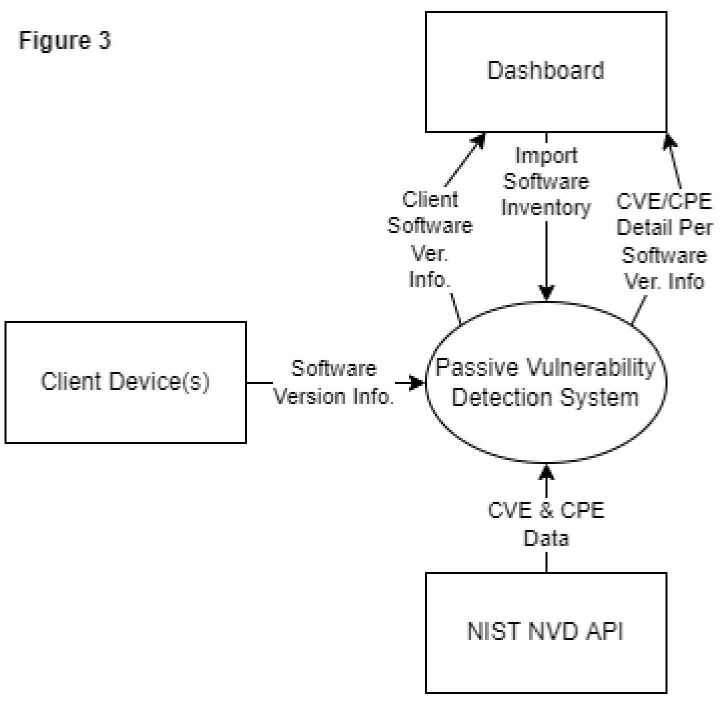


Figure 3
Dashboard
Client Software Ver. Info.
Import Software Inventory
CVE/CPE Detail Per Software Ver. Info
Client Device(s)
Software Version Info.
Passive Vulnerability Detection System
CVE & CPE Data
NIST NVD API

Fig. 3. System Context Diagram

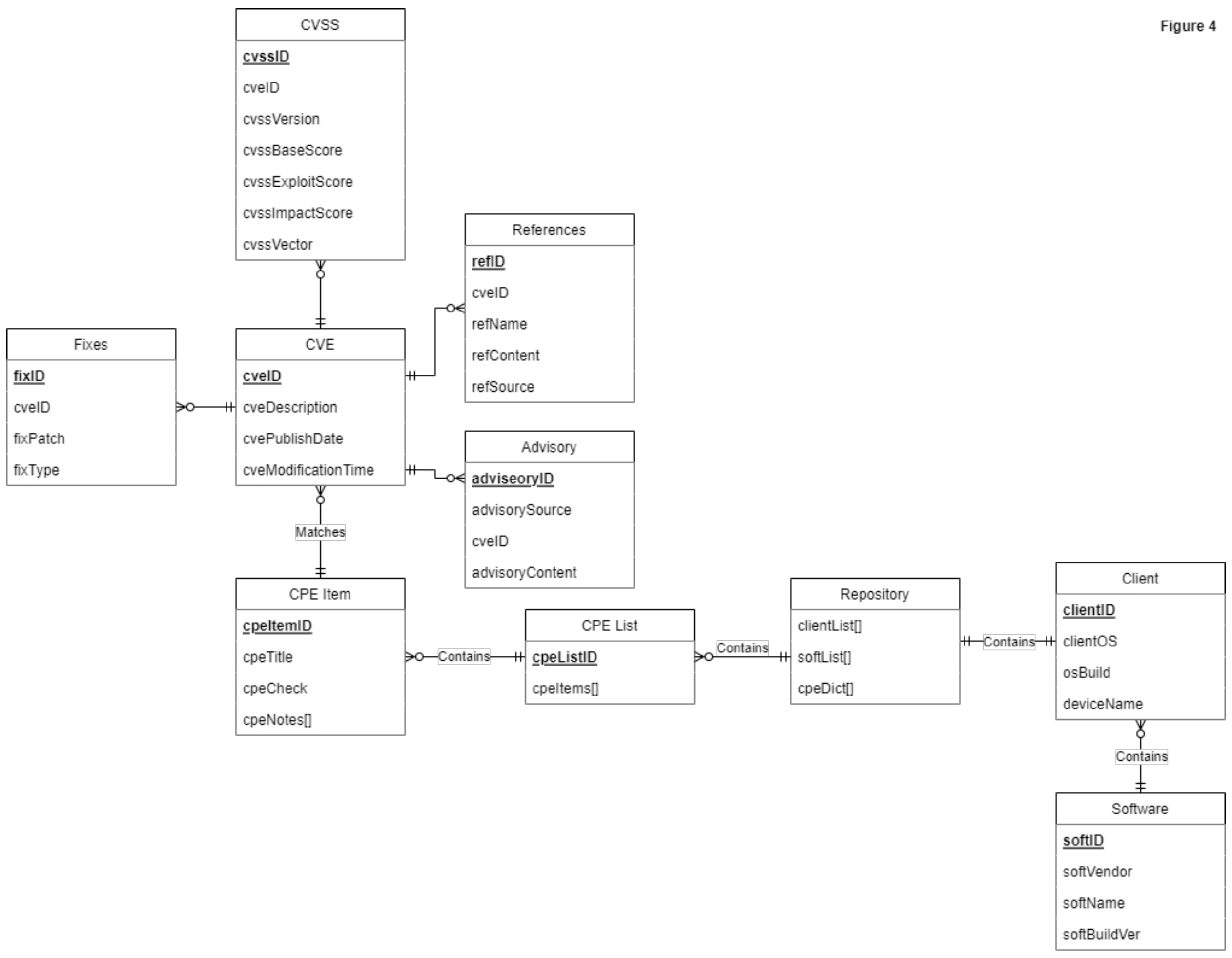


Fig. 4. Entity Relationship Diagram

2.1. String Matching Implementation

As suggested in the research of the Ratcliff/Obershelp formula [5], this algorithm appeared to perform better when matching software to the corresponding CPE identifiers compared to the Levenshtein algorithm. As a result, our main focus for string matching in this Capstone will be on the Ratcliff/Obershelp formula. An example of how we hypothesize we can implement the algorithm is as in Figure 5.

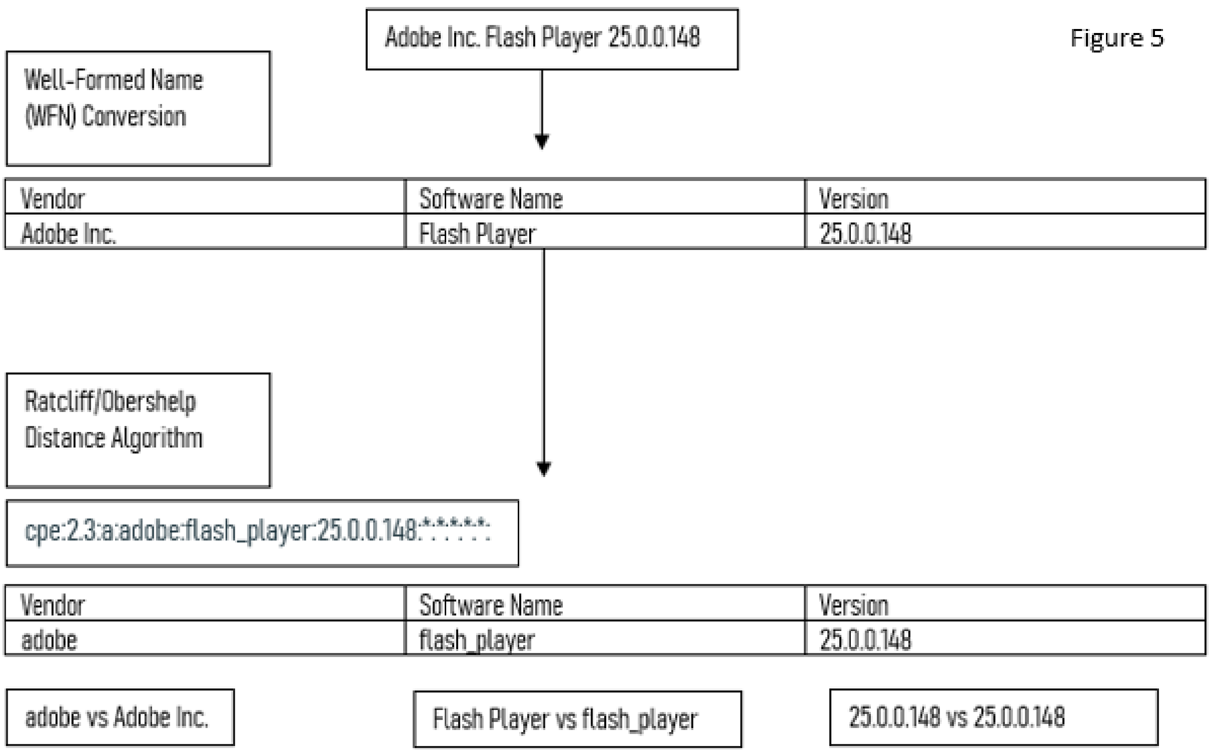


Figure 5
Adobe Inc. Flash Player 25.0.0.148
Well-Formed Name (WFN) Conversion
Vendor
Software Name
Version
Adobe Inc.
Flash Player
25.0.0.148
Ratcliff/Obershelp Distance Algorithm
cpe:2.3:a:adobe:flash_player:25.0.0.148:*:*:*:*:*:
Vendor
Software Name
Version
adobe
flash_player
25.0.0.148
adobe vs Adobe Inc.
Flash Player vs flash_player
25.0.0.148 vs 25.0.0.148

Fig. 5. String Matching Methodology Breakdown

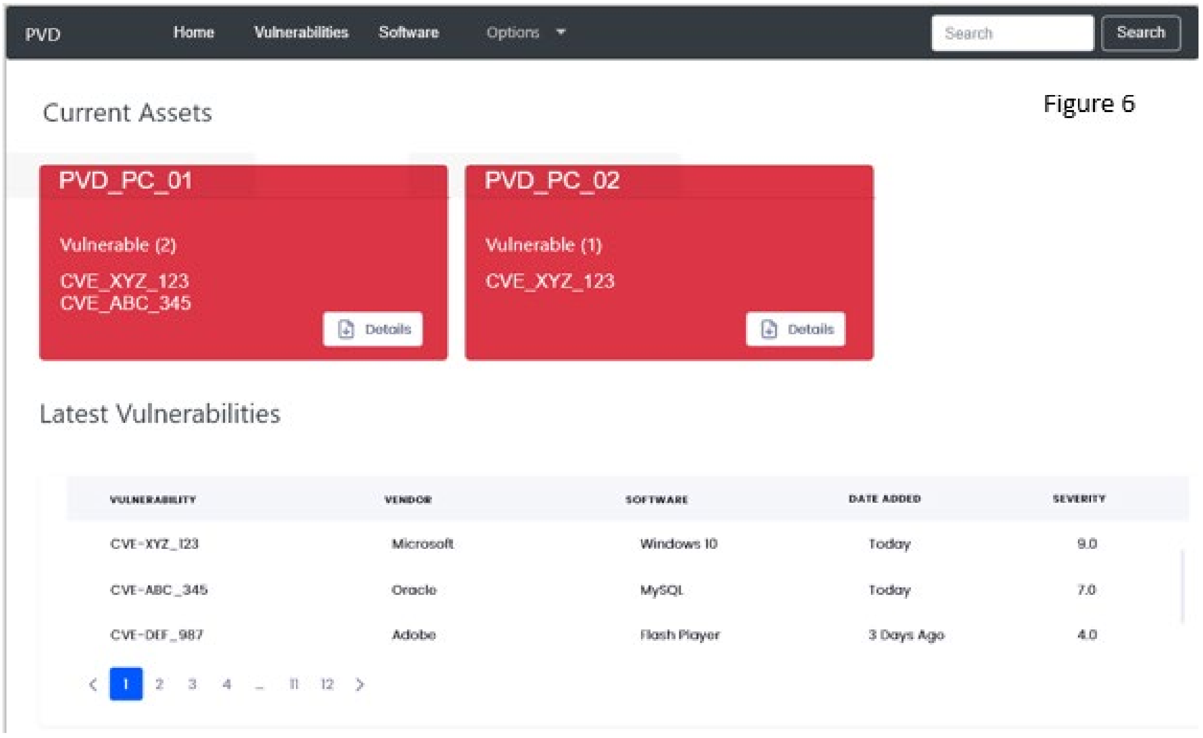
In various cases, the string matching between software descriptions and the CPE will be quite similar if not identical. However, with slight differences in the CPE naming convention, the Ratcliff/Obershelp will be able to aid in identifying the exact matches due to the distance value assigned to the comparison of two strings. As we test the algorithm against various software and their naming conventions, the threshold in which we can deem an accurate match can be tuned.

In addition, the implementation of such a string matching algorithm can be done easily with previously built functions from the Python community. Libraries designed for string matching such as difflib [8] and textdistance [9] can be used to implement the comparison.

2.2. Environment and Testing Setup

In order to test the PVD system, we will be building our own VM image packaged with handpicked software and versions. The first test environment will be a VM image of Windows 10 that will be made to include a sample of both vulnerable and latest/secure versions of software. These software will be chosen on the basis of popularity, recent vulnerabilities and the string matching distance to the related CPE. This way, we can grab a sample of software that encompasses various areas to test for accuracy. It includes known vulnerable versions of applications such as Google Chrome, Apache log4j and more. The vulnerabilities detected can be validated against the documentation for this VM. In addition, another VM built on Debian-Linux will be used in an attempt to compare the validation of vulnerabilities from different environments. This is due to the differences in ways developers and publishers choose to assign descriptors to their softwares in different environments.

2.3. Interface Design



VULNERABILITY	VENDOR	SOFTWARE	DATE ADDED	SEVERITY
CVE-XYZ_123	Microsoft	Windows 10	Today	9.0
CVE-ABC_345	Oracle	MySQL	Today	7.0
CVE-DEF_987	Adobe	Flash Player	3 Days Ago	4.0

Fig. 6. Wireframe Mockup for User Dashboard

The user interface for the PVD will include a dashboard (initial mockup wireframe in Figure 6) setup for administrators to see vulnerabilities on their assets as well as browse vulnerabilities that have been recently added to the NVD regarding software they use. There will also be a details page for each asset and each vulnerability with corresponding metadata/information. The goal is to have a full management system for users to upload software inventory data and have the system passively identify vulnerabilities. Then, the data is aggregated and displayed in the dashboard. Additional features to add include a system to create tickets for remediation, a tracking system for the status of the vulnerabilities and an email alert system.

2.4. Feasibility and Impact

The project that includes the scope of all elements defined in the above figures would need to be completed within the twenty-one weeks allotted. Feasibility of such a task should be achievable given that all deadlines are planned out accordingly and followed. The impact of the proposed solution will be the completion of a passive vulnerability management system that incorporates improved string matching capabilities and support for a large variety of importing data formats.

3. Constraints, Resource Requirements, Risk Assessment

3.1. Constraints

This project being a research project that is expected to be completed during academic terms presents some special constraints that might not be found or expected in other projects of similar nature and scope, while others will be common occurring constraints seen in other projects.

Some identified constraints:

- Time - development time will be limited by the academic terms, therefore there is twenty-one weeks for the planning and development of this research project and its products.
- Skill - members will fill the skill gap as needed as gaps arise through further research and learning.
- OS level & Application Type - where the product for this research project will be located and runs, and what type of application it runs as. This may limit its ease of use for development time.

3.2. Resource Requirements

This project has little resource demand the only possible ones identified are:

- Development & Testing Environment - where development and testing is to occur, as it will affect workflow and processes.
    - A repository for developed parts, code, and collected data sets.
    - Possible Costs - if a cloud based environment is chosen, some costs could be expected, might need to set some time to seek some funding.

3.3. Risk Assessment

There are no real risks that have been identified with this research project's topic and goals, as no dangerous code is expected to be executed or developed nor is the project expected to be tested in a real live environment where it may affect real business processes. This makes the project fairly low-risk.

4. Project Implementation

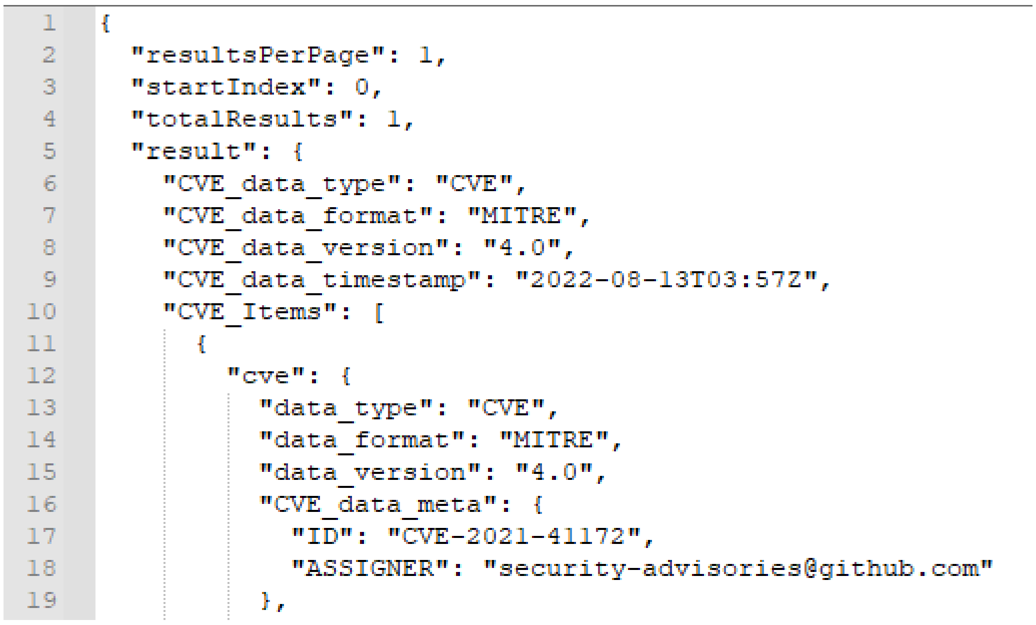
4.1. Background

4.1.1. Vulnerability Data

In the implementation of this project, the NIST NVD Vulnerability Database was chosen to be the primary data source for vulnerability information. The NVD is a United States government database that houses vulnerability management data from the Security Content Automation Protocol (SCAP). The data that the project gathers from the database is Common Vulnerabilities and Exposures (CVE), Common Platform Enumeration (CPE), and Common Vulnerability Scoring System (CVSS). In order to query and store such data, various methods were reviewed and studied. The methods available included storing a local copy of the database, querying 3rd party connections, using exported JSON feeds, and querying the recently added NIST API. The method that was chosen for this project was the API as it provided the most up to date information and removed the need for additional storage.

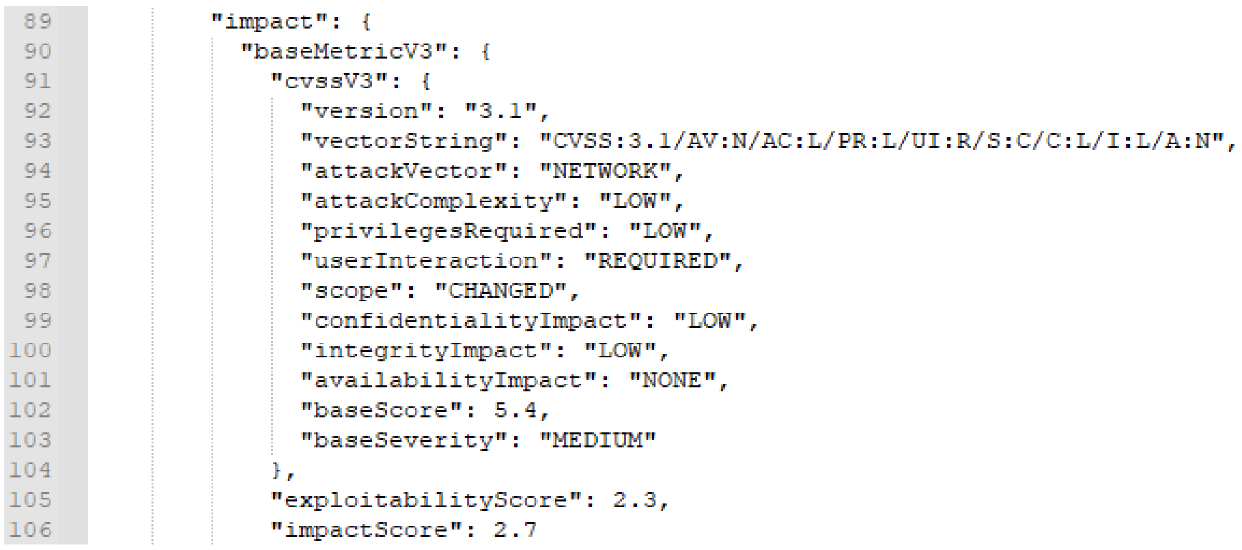
4.1.2. Common Vulnerabilities and Exposures (CVE)

Of the various data fields in the CVE JSON response from the API, the system uses the following fields: CVE ID, CVE Description, CVE Common Weakness Enumeration Specification (CWE) ID, CVSS version 2 base score, and CVSS version 3 base score.



```
{
  "resultsPerPage": 1,
  "startIndex": 0,
  "totalResults": 1,
  "result": {
    "CVE_data_type": "CVE",
    "CVE_data_format": "MITRE",
    "CVE_data_version": "4.0",
    "CVE_data_timestamp": "2022-08-13T03:57Z",
    "CVE_Items": [
      {
        "cve": {
          "data_type": "CVE",
          "data_format": "MITRE",
          "data_version": "4.0",
          "CVE_data_meta": {
            "ID": "CVE-2021-41172",
            "ASSIGNER": "security-advisories@github.com"
          },
```

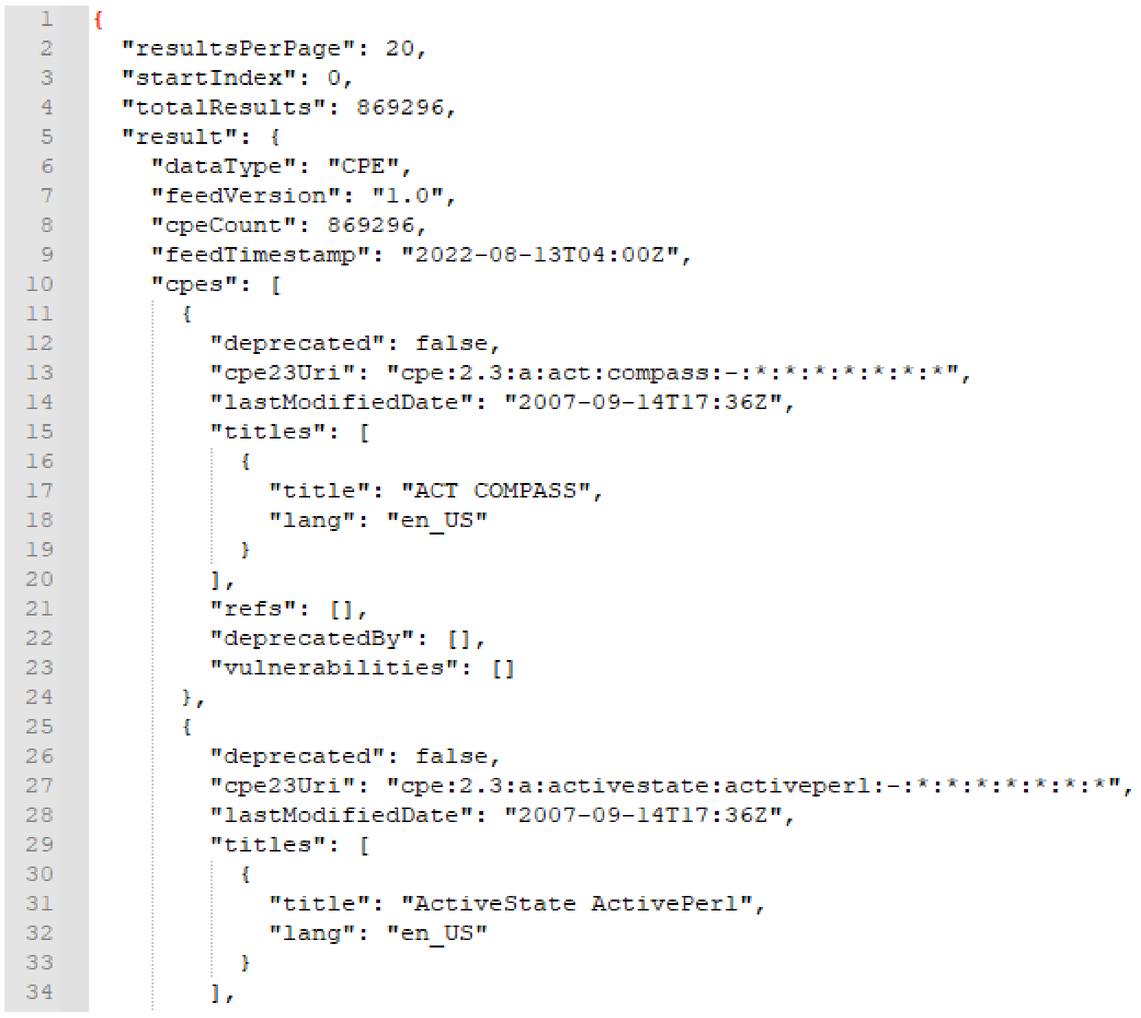
Fig. 7. Common Vulnerability Exposure (CVSS)



```
      "impact": {
        "baseMetricV3": {
          "cvssV3": {
            "version": "3.1",
            "vectorString": "CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:L/A:N",
            "attackVector": "NETWORK",
            "attackComplexity": "LOW",
            "privilegesRequired": "LOW",
            "userInteraction": "REQUIRED",
            "scope": "CHANGED",
            "confidentialityImpact": "LOW",
            "integrityImpact": "LOW",
            "availabilityImpact": "NONE",
            "baseScore": 5.4,
            "baseSeverity": "MEDIUM"
          },
          "exploitabilityScore": 2.3,
          "impactScore": 2.7
```

Fig. 8. Common Vulnerability Scoring System (CVSS)

Of the various fields from the CPE JSON response from the API, the system uses the cpe23Uri field to retrieve the CPE ID.



```
{
  "resultsPerPage": 20,
  "startIndex": 0,
  "totalResults": 869296,
  "result": {
    "dataType": "CPE",
    "feedVersion": "1.0",
    "cpeCount": 869296,
    "feedTimestamp": "2022-08-13T04:00Z",
    "cpes": [
      {
        "deprecated": false,
        "cpe23Uri": "cpe:2.3:a:act:compass:-:*:*:*:*:*:*:*",
        "lastModifiedDate": "2007-09-14T17:36Z",
        "titles": [
          {
            "title": "ACT COMPASS",
            "lang": "en_US"
          }
        ],
        "refs": [],
        "deprecatedBy": [],
        "vulnerabilities": []
      },
      {
        "deprecated": false,
        "cpe23Uri": "cpe:2.3:a:activestate:activeperl:-:*:*:*:*:*:*:*",
        "lastModifiedDate": "2007-09-14T17:36Z",
        "titles": [
          {
            "title": "ActiveState ActivePerl",
            "lang": "en_US"
          }
        ],
```

Fig. 9. Common Platform Enumeration (CPE)

4.1.3. Software Inventory Data

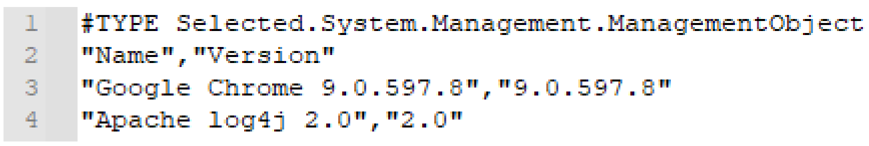
The system uses passively collected software inventory information to feed into the CVE and CPE search functions. There are various ways to gather software inventory information including a CMDB, Software Exports, and scripts to query the operating system. In the case of Windows, PowerShell was used to query system variables to extract installed software. The following command was used to take the software inventory of a Windows system and save as a Comma-separated values (CSV) file:

```
Get-WMIObject -Query "SELECT * FROM Win32_Product Where Not Vendor Like '%Microsoft%'" | Select-Object Name,Version | Export-CSV 'C:\InstallList.csv'
```

A similar command could be used to export all software including default packages from Microsoft:

```
Get-WMIObject -Query "SELECT * FROM Win32_Product" | Select-Object Name,Version | Export-CSV 'C:\InstallList.csv'
```

The output would result in a CSV file with fields for the software name and the version. An example of such an export would result in the following format seen in Figure 10.



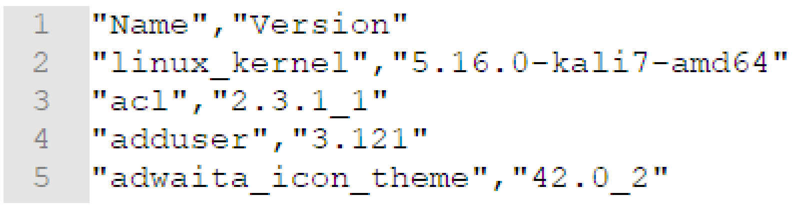
```
#TYPE Selected.System.Management.ManagementObject
"Name","Version"
"Google Chrome 9.0.597.8","9.0.597.8"
"Apache log4j 2.0","2.0"
```

Fig. 10. Sample resulting CSV of software inventory for a Windows system after utilizing the last discussed command in PowerShell

For most Debian-based Linux systems, Terminal was used to query system variables to extract installed software. The following command was used to take the software inventory of a Linux system, including the system's kernel version as an entry, and save as a CSV file:

```
echo '"Name","Version"' > InstallList.csv; echo '"linux_kernel",\c' >> InstallList.csv; uname -r | sed 's/^/"/;s/$/"/' >> InstallList.csv; dpkg-query -W -f='"${Package}","${version}"\n' | tr - _ >> InstallList.csv
```

The output results in a CSV file with similar fields, of that from Windows. An example of such an export would result in the following format seen in Figure 11.



```
"Name","Version"
"linux_kernel","5.16.0-kali7-amd64"
"acl","2.3.1_1"
"adduser","3.121"
"adwaita_icon_theme","42.0_2"
```

Fig. 11. Sample resulting CSV of software inventory for a Linux system after using the previously discussed command in Terminal

4.2. Methodology

4.2.1. String Matching and CPE Search

In order to pass the software from the asset inventory into a proper API query to the NVD, the strings must be verified to follow proper syntax. To achieve this, the software list has already been comma delimited into the following format: *Software Name, Software Version.* However, this is not a catch-all approach as there is no standardized way for Operating Systems and the applications themselves to properly classify themselves in the metadata. For instance, many software may include the version number inside the software name. To do so, regular expression patterns were used to first verify that the software name fit into the following logical flow:

```
If Software Name includes Software Version
        Remove Software Version
Else Combine Software Name and Software Version
Pass combined string to CPE Search
```

This logic was applied to ensure that an accurate string was passed to the CPE API for querying. An example of such a query is as follows:

https://services.nvd.nist.gov/rest/json/cpes/1.0?keyword={cpe_name}

The NVD responds with a JSON list of the 20 most relevant search results for the query. The textdistance python library was then used to apply string matching calculations from the Ratcliff-Obershelp formula to return the best matching result. The result is a correctly formatted CPE string that is then passed to the CVE API for querying.

4.2.2. CVE Search and Retrieval

Using the closest match CPE from the previous step, the string is passed to the CVE API. An example of such a query is as follows:

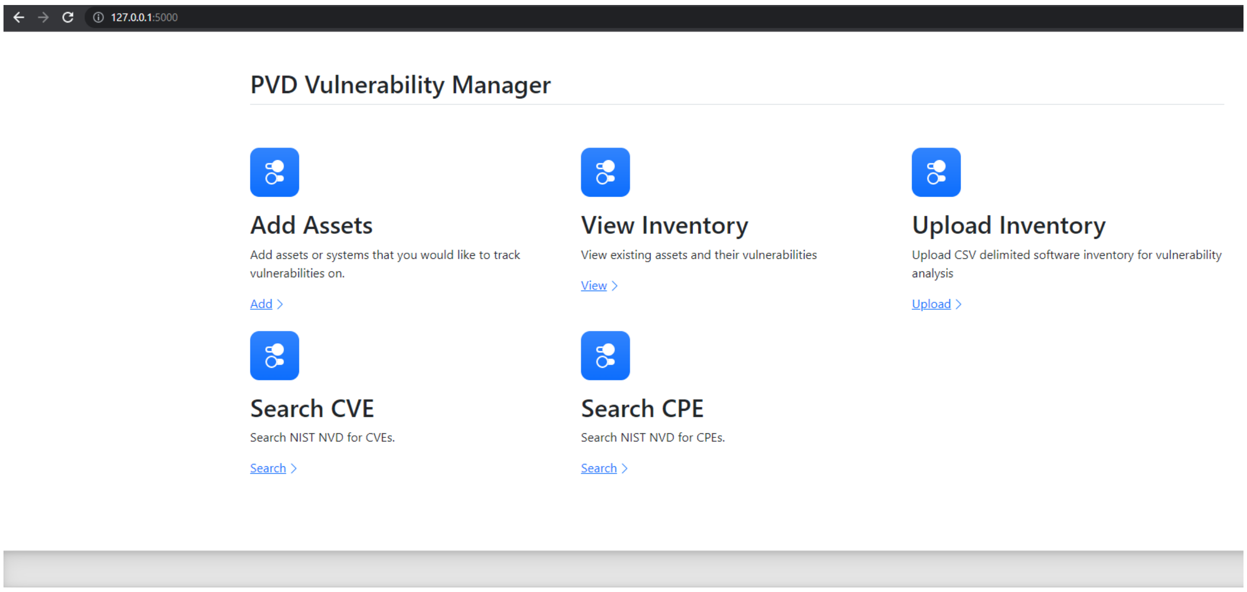
https://services.nvd.nist.gov/rest/json/cves/1.0?cpeMatchString={cpe_id}

The NVD responds with a JSON list of the associated CVE records. These vulnerabilities as well as important metadata fields such as the severity scores are stored and displayed in the system for tracking and reference.

4.3. Design and Testing

4.3.1. System Design

The goal was to include a design that was to offer a very easy to use dashboard for management of Inventory and assets. This was to also include complimentary tools, such as manual search features that could be helpful in quickly verifying any particular piece of software's CVE or CPE available data. The resulting design, could be seen in the Figures


127.0.0.1:5000

PVD Vulnerability Manager

Add Assets

Add assets or systems that you would like to track vulnerabilities on.

Add >

View Inventory

View existing assets and their vulnerabilities

View >

Upload Inventory

Upload CSV delimited software inventory for vulnerability analysis

Upload >

Search CVE

Search NIST NVD for CVEs.

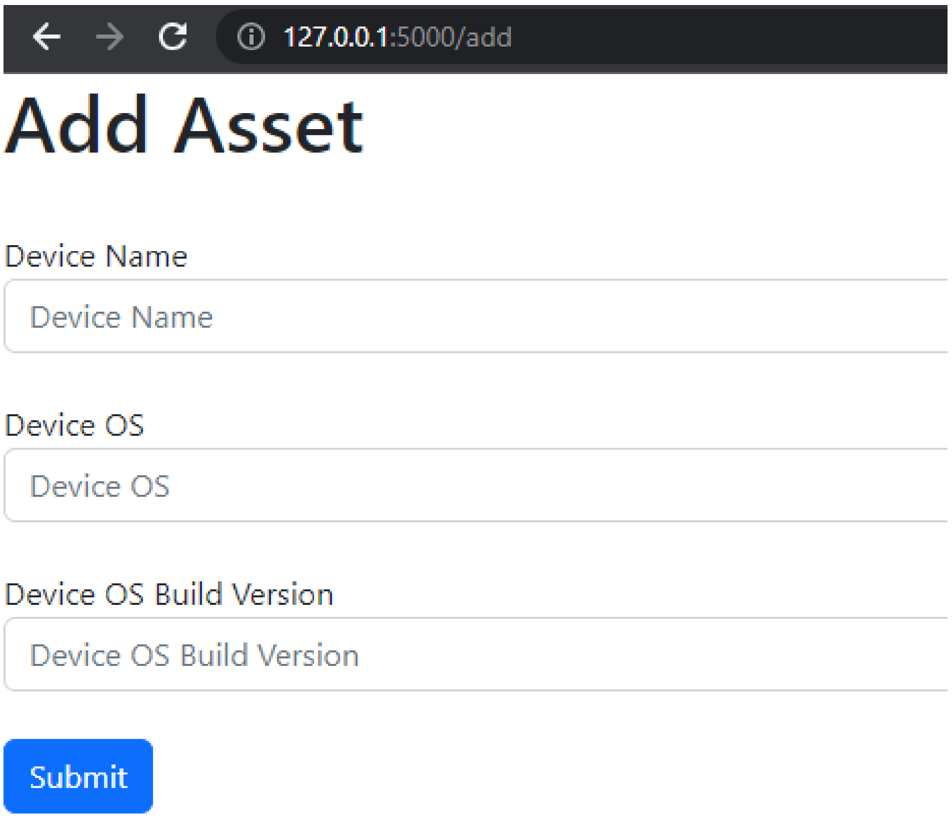
Search >

Search CPE

Search NIST NVD for CPEs.

Search >

Fig. 12. Main System Landing/Dashboard Page


127.0.0.1:5000/add

Add Asset

Device Name

Device Name

Device OS

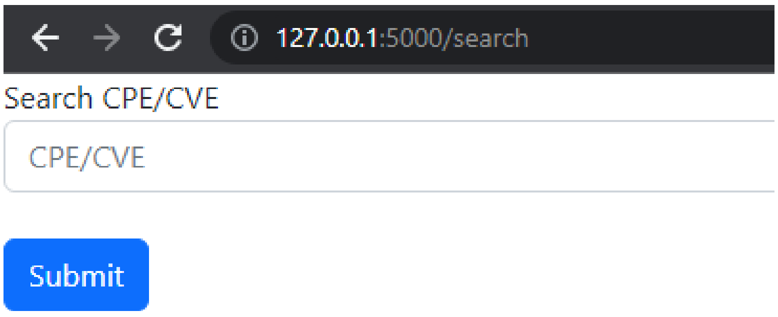
Device OS

Device OS Build Version

Device OS Build Version

Submit

Fig. 13. Adding Additional Asset(s) Option

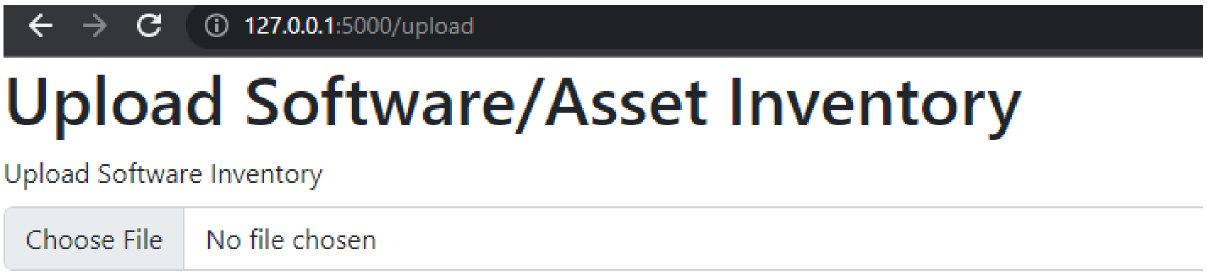

127.0.0.1:5000/search

Search CPE/CVE

CPE/CVE

Submit

Fig. 14. Manual Search Option of CPE/CVE

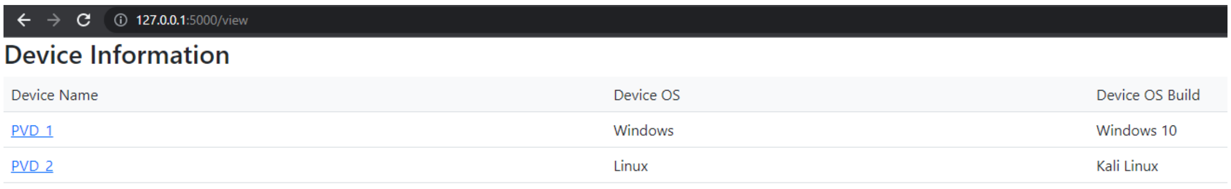

127.0.0.1:5000/upload

Upload Software/Asset Inventory

Upload Software Inventory

Choose File No file chosen

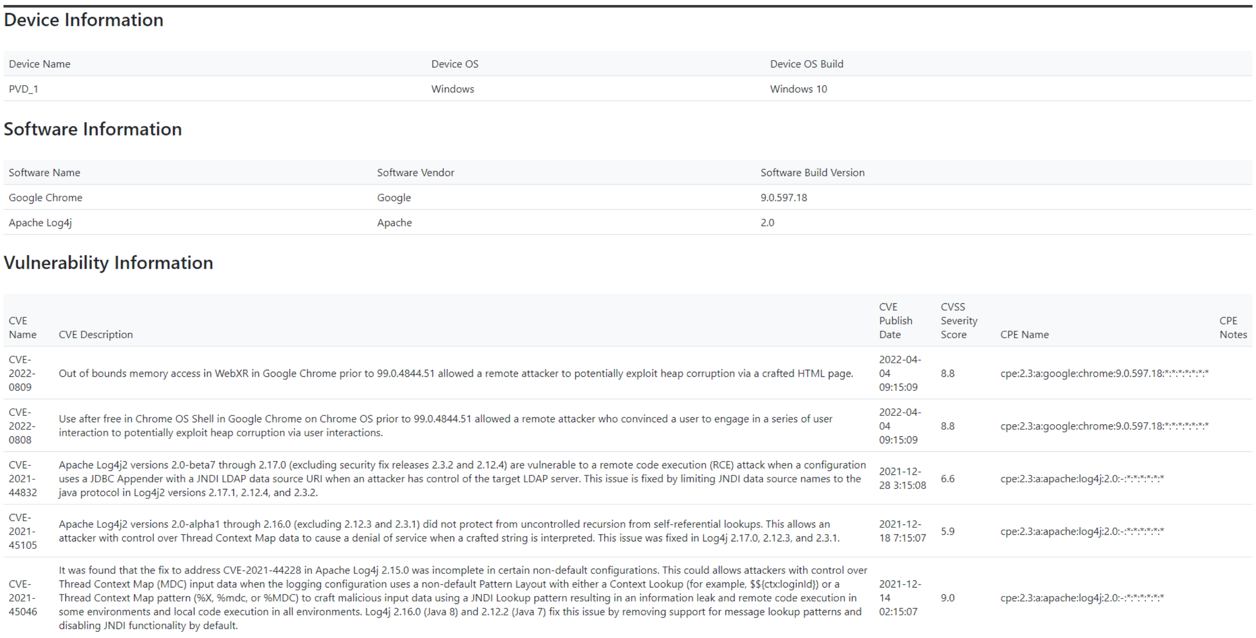
Fig. 15. Tool to Upload additional Software Inventory list for Assets


127.0.0.1:5000/view

Device Information

Device Name	Device OS	Device OS Build
PVD_1	Windows	Windows 10
PVD_2	Linux	Kali Linux

Fig. 16. View of a list of all Assets being Tracked by the System


Device Information

Device Name	Device OS	Device OS Build
PVD_1	Windows	Windows 10

Software Information

Software Name	Software Vendor	Software Build Version
Google Chrome	Google	9.0.597.18
Apache Log4j	Apache	2.0

Vulnerability Information

CVE Name	CVE Description	CVE Publish Date	CVSS Severity Score	CPE Name	CPE Notes
CVE-2022-0809	Out of bounds memory access in WebXR in Google Chrome prior to 99.0.4844.51 allowed a remote attacker to potentially exploit heap corruption via a crafted HTML page.	2022-04-04 09:15:09	8.8	cpe:2.3:a:google:chrome:9.0.597.18:*:*:*:*:*:*:*	
CVE-2022-0808	Use after free in Chrome OS Shell in Google Chrome on Chrome OS prior to 99.0.4844.51 allowed a remote attacker who convinced a user to engage in a series of user interaction to potentially exploit heap corruption via user interactions.	2022-04-04 09:15:09	8.8	cpe:2.3:a:google:chrome:9.0.597.18:*:*:*:*:*:*:*	
CVE-2021-44832	Apache Log4j2 versions 2.0-beta7 through 2.17.0 (excluding security fix releases 2.3.2 and 2.12.4) are vulnerable to a remote code execution (RCE) attack when a configuration uses a JDBC Appender with a JNDI LDAP data source URI when an attacker has control of the target LDAP server. This issue is fixed by limiting JNDI data source names to the java protocol in Log4j2 versions 2.17.1, 2.12.4, and 2.3.2.	2021-12-28 3:15:08	6.6	cpe:2.3:a:apache:log4j:2.0:-:*:*:*:*:*:*	
CVE-2021-45105	Apache Log4j2 versions 2.0-alpha1 through 2.16.0 (excluding 2.12.3 and 2.3.1) did not protect from uncontrolled recursion from self-referential lookups. This allows an attacker with control over Thread Context Map data to cause a denial of service when a crafted string is interpreted. This issue was fixed in Log4j 2.17.0, 2.12.3, and 2.3.1.	2021-12-18 7:15:07	5.9	cpe:2.3:a:apache:log4j:2.0:-:*:*:*:*:*:*	
CVE-2021-45046	It was found that the fix to address CVE-2021-44228 in Apache Log4j 2.15.0 was incomplete in certain non-default configurations. This could allows attackers with control over Thread Context Map (MDC) input data when the logging configuration uses a non-default Pattern Layout with either a Context Lookup (for example, $${ctx:loginId}) or a Thread Context Map pattern (%X, %mdc, or %MDC) to craft malicious input data using a JNDI Lookup pattern resulting in an information leak and remote code execution in some environments and local code execution in all environments. Log4j 2.16.0 (Java 8) and 2.12.2 (Java 7) fix this issue by removing support for message lookup patterns and disabling JNDI functionality by default.	2021-12-14 02:15:07	9.0	cpe:2.3:a:apache:log4j:2.0:-:*:*:*:*:*:*	

Fig. 17. Detailed view of an Asset, displaying all the matched relevant CVE information for the Inventory

4.3.2. Environment and Testing

The Windows Environment used to test the system was built in a Windows 10 VM using VMWare. To test the system specifically vulnerable software versions were chosen and installed on the VM. This included vulnerable versions of the Google Chrome web browser 9.0.597.18, Apache Log4j 2.0, Oracle MySql 5.7.1, and Adobe Flash Player 26.0.0.137. The following powershell script was used to collect the software inventory from the system:

*Get-WMIObject -Query "SELECT * FROM Win32_Product Where Not Vendor Like '%Microsoft%'" | Select-Object Name,Version | Export-CSV 'C:\InstallList.csv'*

Filtering the list, the CSV file was passed to the system to test if it could correctly identify the CPE for the 4 vulnerable software and the associated vulnerabilities for each one. The results were as in Table II, that follows.

TABLE II
Returned CPE Accuracy for Software WFNs

Software WFN	Returned CPE Identifier	Correct CPE	Vulnerability Detection
Google Chrome 9.0.597.18	cpe:2.3:a:google:chrome:9.0.597.18:*:*:*:*:*:*:*	Yes	Accurate
Apache Log4j 2.0	cpe:2.3:a:apache:log4j:2.0:-:*:*:*:*:*:*	Yes	Accurate
Oracle MySql 5.7.1	cpe:2.3:a:oracle:mysql:5.7.1:*:*:*:*:*:*:*	Yes	Accurate
Adobe Flash Player 26.0.0.137	cpe:2.3:a:adobe:flash_player:26.0.0.137:*:*:*:*:*:*:*	Yes	Accurate

For the supplied WFN identifiers, the system was able to query the correct CPE identifier through the CPE API and string matching. With the correct CPE identifier all related vulnerabilities indexed in the NVD were also correctly queried.

4.4. Discussion

4.4.1. Issues

One of the main issues observed during the development of the system was the flaws of the CPE system. Despite the flaws, the CPE and CVE system from NIST is still the most efficient and standardized system to categorize software and their vulnerabilities. The first issue is the mapping between CVE and CPE entries. NIST performs all the validation and mapping on their end but there are sometimes missing relationships between these data structures. Certain vulnerabilities may not be mapped to a specific CPE due to no CPE existing at the time. According to other researchers, 895 CVE entries are not linked to a specific CPE [4]. In addition, CPE identifiers may not exist for every product or software. NIST does not automatically add CPE identifiers to all software observed. Thus, if the system were to search for a software within the inventory that does not have a CPE identifier, the results would be inaccurate. This is something that through this project was found to affect many Linux softwares. Also, if an organization used proprietary software developed in-house, vulnerabilities that could be detected with signatures of active scanning may not be detectable with a passive system. Lastly, the CPE naming convention has gone through different iterations. Certain identifiers may have different spelling which can lead to once again inaccurate search results.

On the other hand, issues that were previously considered such as synchronization and relative accuracy from string matching have been resolved in this system. The synchronization with the NIST database is maintained through the use of the API directly querying the live copy. This removes any problems that a local copy or static file may pose. In addition, the two-folded approach of running an initial query of the CPE dictionary and performing additional string matching validation has allowed the system to conclude the most relevant CPE identifier for a specific software WFN.

4.4.2. Improvements and Future Work

Expanded coverage of software inventory formats is one major improvement for the system. Currently, the accepted format is a CSV file exported using operating system level scripting languages. However, there are many additional methods to gather software inventories including CMDBs and third party software. Due to limitations, these avenues were lightly and unsuccessfully explored for this project. Expanding the supported formats would allow for even more organizations and individuals to make use of this passive vulnerability management system.

In addition to this, the system has only the mandatory components of a passive vulnerability management framework. There are additional areas to explore such as alerting and ticketing. Integrating periodic queries of the NIST NVD database to verify if uploaded software contains new vulnerabilities could allow for real time alerting. In addition, integrating a ticketing system to resolve detected vulnerabilities through systems like JIRA can also be implemented to streamline the process for vulnerability remediation.

References

[1] T. Palmaers, “Implementing a Vulnerability Management Process”, SANS Whitepaper, 2013 [Online]. Available: https://www.sans.org/white-papers/34180/

[2] H. Ecik, "Comparison of Active Vulnerability Scanning vs. Passive Vulnerability Detection," 2021 International Conference on Information Security and Cryptology (ISCTURKEY), 2021, pp. 87-92, doi: 10.1109/ISCTURKEY53027.2021.9654331 [Online]. Available: https://ieeexplore-ieee-org.library.sheridanc.on.ca/document/9654331

[3] National Vulnerability Database, “Vulnerabilities”, National Institute of Standards and Technology. [Online]. Available: https://nvd.nist.gov/vuln. [Accessed Jan. 28, 2022]

[4] L. A. B. Sanguino and R. Uetz, “Software vulnerability analysis using CPE and CVE”, arXiv preprint arXiv:1705.05347.5), 2017 [Online]. Available: https://arxiv.org/abs/1705.05347

[5] R. Ushakov, E. Doynikova, E. Novikova and I. Kotenko, "CPE and CVE based Technique for Software Security Risk Assessment," 2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2021, pp. 353-356, doi: 10.1109/IDAACS53288.2021.9660968 [Online]. Available: https://ieeexplore-ieee-org.library.sheridanc.on.ca/document/9660968

[6] D. Tovarňák, L. Sadlek and P. Čeleda, "Graph-Based CPE Matching for Identification of Vulnerable Asset Configurations," 2021 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2021, pp. 986-991 [Online]. Available: https://ieeexplore-ieee-org.library.sheridanc.on.ca/document/9463994

[7] E. Wåreus and M. Hell, “Automated CPE Labeling of CVE Summaries with Machine Learning,” Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 3–22, 2020, doi: 10.1007/978-3-030-52683-2_1 [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-52683-2_1

[8] B. Peterson, R. Hettinger, et al. difflib. Python Core Library Package. 2009. Available: https://python.readthedocs.io/fr/stable/library/difflib.html

[9] N.G. Voronov, et al. Textdistance. Python package. 2017. Available: https://github.com/life4/textdistance