Analysis of CISA's KEV To Help Future Exploitation Defense

Tristan Freeman

A Capstone Project Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science
Congdon School of Supply Chain, Business Analytics and Information Systems

University of North Carolina Wilmington

2023

Approved by

Advisory Committee

_____	_____
Barry Wray	Hosam Alamleh

_____
Geoff Stoker, Chair

Accepted By

_____
Dean, Graduate School

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Analysis of CISA's KEV To Help Future Exploitation Defense. Freeman, Tristan, 2023. Capstone Paper, University of North Carolina Wilmington.

Every year there are more Common Vulnerabilities and Exposures (CVE) published in the National Vulnerability Database (NVD) than the year before, yearly reports have now surpassed more than 20,000 reports a year, but the most important CVEs are the ones that are exploited. Specifically exploited CVEs are recorded by the Cybersecurity and Infrastructure Security Agency (CISA) and placed into their Known Exploited Vulnerabilities Catalog (KEV). The focus of this research is to analyze commonalities of CVEs from the KEV and the NVD and create predictions from the data spanning from 2003 to 2023. I extracted all 200,000+ CVEs from the NVD and all 1,000+ CVEs from the KEV and deleted all Common Vulnerability Scoring System (CVSS) that used only version 3 to eliminate possible outliers. Using three different approaches to predict KEV CVEs I found that it is best to predict future attacks using machine learning because we're able to predict 80% of CVEs that could end up being part of the KEV using common variables between NVD CVEs and KEV CVEs. The findings from this capstone can help assist cybersecurity professionals in preventing attacks by telling them which CVEs need to be patched and which do not.

Keywords- CVE; NVD; CISA; KEV; CVSS

CHAPTER 1: INTRODUCTION

The number of published CVE has gone from below 10,000 a year between 2002 and 2015 to over 20,000 every year from 2020 to the present. CVEs are publicly disclosed security flaws, and sometimes these published flaws are exploited. That is where the CISA steps in and adds it to their catalog for KEV, this catalog specifically records exploited CVEs. This capstone will outline the research and analysis of taking those CVEs from within CISA's KEV and comparing them to the National Institute of Standards and Technology's (NIST) NVD, a database of all published security flaws. From there, research will be able to determine ways cybersecurity professionals can defend against CVEs. This is a huge task because the NVD holds over 200,000 CVEs while the KEV only holds a little over 1,000. From looking at the data, it is apparent that most CVEs are never actually exploited, and one can easily assume that it is difficult for cybersecurity professionals to know which CVEs should get priority in remediation and mitigation efforts. This research will discuss different approaches that cybersecurity professionals can utilize to address this issue by comparing different factors of CISA's KEV and NIST's NVD to make predictions on CVEs likely to be exploited.

The data needs to be analyzed and configured to figure out what resolutions are possible to resolve any incoming CVEs. In 2022, there were 25,101 published NVD CVEs, which equates to nearly 69 CVEs a day across 365 total days in year. However, this calculation is inaccurate because it spans over the course of all days in a year and cybersecurity professionals likely do not work all 365 days in a year. Assuming, on average, that a CVE can be read, understood, then discarded or actioned within an hour, it would take a staff of thirteen employees working normal eight-hour shifts dedicated to CVE mitigation to manage this massive workload. Small or midsized businesses with

eight or fewer employees dedicated to CVE management would not be able to keep up. This research looks to better equip cybersecurity professionals to manage the CVE onslaught by using different approaches, such as machine learning, to predict which CVEs are likely to be exploited by malicious actors and should be patched before an attack occurs and which CVEs can be ignored.

Every year there is a gradual growth in the number of CVEs published and recorded, attackers are adapting and growing just as much as cybersecurity professionals. Suggestions on how to possibly deter future attacks are needed, to help slow the rate of future attacks. This project seeks to find helpful suggestions for mitigating future attacks by studying data recorded from past CVEs to track similarities and determine how the incident could have been avoided based on those similarities. This research hopes to find actionable suggestions that make it possible to predict CVEs from the similarities between each one and inform cybersecurity professionals on potential KEV CVEs.

If cybersecurity professionals can utilize this research to prevent or avoid some future attacks, it could potentially save organizations unnecessary expenditure of scarce resources, primarily their time and money. The research stems from data extracted from public resources that NIST and CISA have published on their websites. The objective is to take the CVE IDs from the KEV then programmatically search those IDs within the NVD API and place any necessary data into a spreadsheet that can be analyzed. Once the entire document is compiled, it is important that it is kept up to date because the number of NVD CVEs and KEV CVEs are both gradually growing daily. The research must be accurate and once the data is ready to be analyzed, no more updates can be made to the data. After the freeze on data collection the data has to be revised and finalized and the data will be analyzed, and conclusions can be made.

CHAPTER 2: REVIEW OF LITERATURE REVIEW AND ANALYSIS

2.1 Analysis of 26000+ CVEs in 2022

This web article covers researchers' analysis of CVEs from 2018-2022 which spanned over 26,000 C. From their analysis, they were able to make charts using Pandas in Python for their data visualization charts. They used these charts to compare their findings by covering common attack vectors, common CWE IDs, and which company had the most CVEs published about them (Figure 1). To gather their research, they used NIST's NVD API to pull data from CVEs and placed their findings into a JSON file, after all their findings were placed into a JSON file, they created a Python script to parse through the raw JSON data to created human-readable fields for each vulnerability entry in the database.

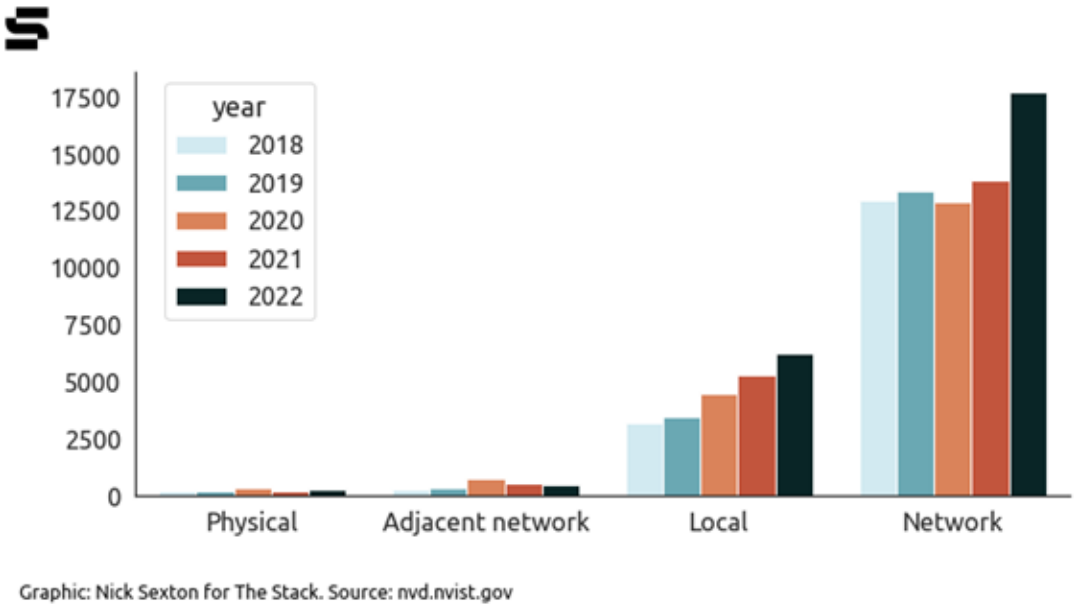

Figure 1. Chart of attack vectors from 2018-2022 (Stack)

This analysis is the exact same outcome this project looks to find; however, their research specifically covers CVEs from 2016 to 2022 and they did not discuss how future attacks can be avoided using the data they found. Their research seems to have no goal

but to bring the issue of the always-rising number of CVEs that are recorded each year into the light. The goal of this research project is to not only bring the issue of rising CVEs into the light so others can realize how dangerous CVEs are but also to analyze the found data and create predictions on what CVEs could end up being exploited and added to the KEV list. Cybersecurity professionals will then be informed of how they can prepare themselves for future attacks against potential KEV CVEs. Data collection for this project will also utilize the NVD API and a Python script to parse through JSON data to create a collection of data that can then be analyzed.

2.2 Trend Analysis of the CVE for Software Vulnerability Management

This research paper was written by four researchers on the information systems security management team from Concordia University College of Alberta in Canada, together they covered all CVEs from 2007 to 2010 equating to over 22,521 CVEs analyzed. The main source they used to compare their findings was the common vulnerability scoring system (CVSS) from the NVD. From the research they found from using the CVSS to compare data, they were able to create a table of fourteen common vulnerabilities previously covered by another author and then create a fifteenth vulnerability from their own research because it was the most difficult attack to detect with only 0.1% of findings being reported in 2006 (Table 1). From the analysis of these specific fifteen vulnerabilities, they recorded many trends and hypothesized that with this data they found information security professionals could be informed and with this information they could perfect their efforts on preventing and mitigating the impact of attacks. Table 2 is another chart they used for data visualization that lists off all of their fifteen vulnerability types ranking them by severity.

This work also relates to the outcome of this project because they were also

hypothesizing that with their research, they could inform cybersecurity professionals on how to reduce future attacks from occurring. The only downside to their research is that it only covers from 2007-2010 so more research must be conducted to keep the trend analysis up to date. This research project will utilize a tool they did not have back then, CISA's KEV catalog. With this catalog, research will be able to focus on preventing the exploitation of CVEs by analyzing CVEs that are recorded as being previously exploited. This tool was not available to them because it was published in 2021 so it was not around when their paper was written.

Table 1

Frequency Order of 15 Vulnerability Types

	Years	2007	2008	2009	2010	Total
Rank	Frequency / Vulnerability	6516	5632	5733	4640	22521
[1]	SQL injection	(4) 687 10.54%	(1) 1090 19.35% 56.66%	(1) 948 16.54% -13.03%	(4) 515 11.10% -45.67%	3240
[2]	Cross-Site Scripting (XSS)	(1) 824 12.65%	(2) 790 14.03% -4.13%	(2) 821 14.32% 3.92%	(2) 594 12.80% -27.65%	3029
[3]	Denial of Service	(3) 793 12.17%	(3) 597 10.60% -24.72%	(3) 689 12.02% 15.41%	(1) 678 14.61% -1.60%	2757
[4]	Buffer overflow	(2) 812 12.46%	(4) 576 10.23% -29.06%	(4) 571 9.96% -0.87%	(3) 541 11.66% -5.25%	2500
[5]	Privilege action	208 3.19%	(5) 440 7.81% 111.54%	(5) 437 7.62% -0.68%	(5) 350 7.54% -19.91%	14358
[6]	Directory traversal	340 5.22%	353 6.27% 3.82%	319 5.56% -9.63%	273 5.88%- 14.42%	1285
[7]	PHP remote file inclusion	(5) 684 10.50%	154 2.73% -77.49%	128 2.23% -16.88%	72 1.55% -43.75%	1038
[8]	Information leak/disclosure	172 2.64%	222 3.94% 28.32%	216 3.77% -2.70%	222 4.78% -2.70%	832
[9]	Authentication	172 2.64%	173 3.07% 57.27%	266 4.64% 53.76%	119 2.56% -55.26%	668

	Years	2007	2008	2009	2010	Total
[10]	Integer overflow	109 1.67	102 1.81% -6.42%	118 2.06% 15.69%	120 2.59% 1.69%	449
[11]	Race condition	67 1.03%	177 3.14% 164.18%	58 1.01% -67.23%	53 1.14% -8.62%	355
[12]	Cross-Site Request Forgery (CSRF)	63 0.97%	76 1.35% 20.63%	113 1.97% 48.68%	76 1.64% -32.74%	328
[13]	Cryptographic error	53 0.81%	59 1.05% 11.32%	110 1.92% 86.44%	90 1.94% -18.18%	312
[14]	Format string	67 1.03%	30 0.53% -55.22%	26 4.45% -13.33%	15 0.32% -42.31%	138
[15]	CRLF injection	34 0.52%	10 1.18% -70.59%	11 1.19% 10.00%	5 0.11% -54.55%	60
	Amount CVEs of 15 types in each year	5023	4849	4831	3723	18427
	Percentages of 15 types in each year	77.10%	89.10%	84.27%	80.24%	81.82%

(Chang, Zavarsky, Ruhl, Lindskog).

Table 2

Severity Order of 15 Vulnerability Types

Rank	Years / Vulnerability	2007	2008	2009	2010	Average
[1]	Buffer overflow	7.98	8.42	8.54	8.44	8.35
[2]	Integer overflow	7.65	7.06	8.01	7.97	7.92
[3]	Format string	7.36	7.15	7.73	7.58	7.46
[4]	PHP remote file inclusion	7.45	7.76	7.26	7.13	7.41
[5]	SQL injection	7.43	7.38	7.36	7.43	7.40
[6]	Authentication	7.46	7.4	6.97	6.79	7.16
[7]	Directory traversal	6.48	6.49	6.49	6.44	6.48

Rank	Years / Vulnerability	2007	2008	2009	2010	Average
[8]	Denial of Service	5.97	6.15	6.13	6.52	6.19
[9]	Privilege action	6.18	6.59	6.16	5.78	6.18
[10]	Cross-Site Request Forgery (CSRF)	5.8	5.72	6.62	6.39	6.13
[11]	CRLF injection	6.91	5.34	6.64	3.94	5.71
[12]	Race condition	5.48	6.38	5.7	5.09	5.66
[13]	Cryptographic error	5.36	5.36	6.05	5.67	5.61
[14]	Information leak/disclosure	5.48	4.97	4.87	4.35	4.92
[15]	Cross-Site Scripting (XSS)	4.66	4.23	4.21	4.13	4.31

(Chang, Zavarsky, Ruhl, Lindskog)

CHAPTER 3: METHODOLOGY

3.1 Implementation

This research will take all of the KEV CVE IDs and extract the full report for each ID from the NVD. While the data within the KEV could be extracted, it is best to use the data from the NVD because the KEV does not have as thorough information as the NVD such as CVSSv2 scores which are going to be the key aspect of the analysis. Later on, we will extract all CVEs that only have CVSSv3 in them, which mostly includes the most recently published CVEs. We must focus on only analyzing CVSSv2 because the bulk of the CVE data has CVSSv2 scores, some CVEs have both CVSSv2 and CVSSv3 and some only have CVSSv3, but this research will focus on the majority of data rather than the most recent.

The difference in information recorded in the KEV records compared to the NVD records is very vast. The KEV records only report on the ID, the affected company, the affected product, and a description of the attack. This information is minimal compared to what is contained within the NVD. The information recorded in the NVD consists of many different types of information such as the severity, exploitability, and impact of the attack and most importantly the CVSSv2 scores. With this extra information, it will greatly assist in feeding the machine learning algorithm more sample data to learn from when analysis is executed allowing a greater opportunity of predicting future cybersecurity breaches and preventing CVEs.

To execute this research, the databases from NIST and CISA must be accessed and utilized, as previously stated. The following databases have grown significantly since this research began in February 2023. For example, the total number of reported CVEs in the KEV was 887 when this project was first started. The number of reported CVEs has

now grown to 996 by September 2023. Similarly, the total number of CVEs within the NVD, in February 2023 there was 195760 and by September 2023, there is now 217254. This growth of data creates an extra amount of work for this research because the documentation will need to be continually updated whenever a new CVE is added to the KEV, it is important to ensure that the data is accurate. These large data sets could prove to be very helpful in resolving and finding new similarities within each CVE. The updating of research will eventually come to a halt with no further updates so that the data can be analyzed, for this instance, data was brought to a halt in September 2023.

3.2 Analysis

To begin this analysis, the yearly JSON files that compile the information of all NVD CVEs must first be downloaded, for this research, any CVEs before 2003 are being excluded with an exception to CVE-2002-0367, which is part of the KEV. The NVD data is needed to create the JSON file that will hold all 200000+ NVD CVEs so a Python file can randomly pick from the batch and allow a machine learning program to analyze and learn from the sample data. With this sample data, the machine can take in the differentiation between each KEV CVE and NVD CVE and predict how to tell them apart and if it is possible to predict which CVE could potentially be exploited to make it a KEV. The most important source of data extraction will be from utilizing the NVD API, a Python file must be created that can look up each KEV CVE within the NVD API individually by their CVE ID. The algorithm must check if the URL status is equal to 200 meaning it can successfully reach the URL and then the JSON data must be loaded into a Python dictionary, dumped into a JSON string, and then printed (Figure 2).

```
import requests
import json

print("Starting.  Trying to contact NIST API...")

url = "https://services.nvd.nist.gov/rest/json/cve/1.0/"

c1 = "CVE-2023-38205"

rQ = requests.get(url+c1)

if rQ.status_code == 200:
    pQ = json.loads(rQ.text)
    print(json.dumps(pQ,indent=2))
    #print()

#END
```

Figure 2. Example of Python File for NVD API Extraction.

This code can access the specific URL for each CVE using the API. The API holds all necessary information for all published CVEs that are needed to feed sample data to the machine learning program. To maximize efficiency, CVE IDs can be extracted from the API in intervals of five using the Python file, if more than five requests are executed the Python program would not completely execute the algorithm and print all requests. The KEV CVE data from the URL will come out in JSON format and this data can be pasted into a new JSON file to compile all entries into one place. However, before pasting the data it must first be altered so that all CVEs are placed into a singular list that can be easily parsed through with another Python file once it is all compiled. To do this, the first 11 lines and final three lines must be deleted after the first CVE is pasted into the document, the data printed from the API has the instance of CVE_Items with every CVE but it is only necessary that this list is established once and all data after the first CVE is copied starting with the line the 12th line so another instance is never created. The final three lines are also detrimental to the data parsing because those lines pertain to closing the list and other variables that are created in the first 12 lines all that is needed to

separate each CVE after the first one is pasted is a singular comma and an open parenthesis. Once all KEV data is compiled into a singular JSON file correctly another Python program must be written to parse through the CVE_Items list and print specific variables necessary for sample data needed for the machine learning program (Figure 3). Once extracted, these specific variables can be pasted into a CSV file giving each CVE its row and all necessary data needed for each CVE its columns (Figure 4).

```
import json

foi = "CVE_Capstone.json"
fp = open(foi, 'r', encoding="utf-8")
data = json.load(fp)
fp.close()

for cve in data["CVE_Items"]:
    cveID = cve["cve"]["CVE_data_meta"]["ID"]

    #print(cveID)

    if "baseMetricV2" in cve["impact"].keys():
        accessVector2 = cve["impact"]["baseMetricV2"]["cvssV2"]["accessVector"]
        accessComplexity2 = cve["impact"]["baseMetricV2"]["cvssV2"]["accessComplexity"]
        authentication2 = cve["impact"]["baseMetricV2"]["cvssV2"]["authentication"]
        confidentialityImpact2 = cve["impact"]["baseMetricV2"]["cvssV2"]["confidentialityImpact"]
        integrityImpact2 = cve["impact"]["baseMetricV2"]["cvssV2"]["integrityImpact"]
        availabilityImpact2 = cve["impact"]["baseMetricV2"]["cvssV2"]["availabilityImpact"]
        baseScore2 = cve["impact"]["baseMetricV2"]["cvssV2"]["baseScore"]
        severity2 = cve["impact"]["baseMetricV2"]["severity"]
        exploitabilityScore2 = cve["impact"]["baseMetricV2"]["exploitabilityScore"]
        impactScore2 = cve["impact"]["baseMetricV2"]["impactScore"]
        obtainAllPrivilege = cve["impact"]["baseMetricV2"]["obtainAllPrivilege"]
        obtainUserPrivilege = cve["impact"]["baseMetricV2"]["obtainUserPrivilege"]
        obtainOtherPrivilege = cve["impact"]["baseMetricV2"]["obtainOtherPrivilege"]

        #print(accessVector2)
        #print(accessComplexity2)
        #print(authentication2)
        #print(confidentialityImpact2)
        #print(integrityImpact2)
        #print(availabilityImpact2)
        #print(baseScore2)
        #print(severity2)
        #print(exploitabilityScore2)
        #print(impactScore2)
        #print(obtainAllPrivilege)
        #print(obtainUserPrivilege)
        #print(obtainOtherPrivilege)
        #print(cveID, " ")
        pass

    else:
        print(cveID)
        pass


#END
```

Figure 3. Example of Python Code for Pulling JSON Values

cveID	accessVector	accessComplexity	authentication	confidentialityImpact
CVE-2002-0367	LOCAL	LOW	NONE	COMPLETE
CVE-2004-0210	LOCAL	LOW	NONE	COMPLETE
CVE-2004-1464	NETWORK	LOW	NONE	NONE
CVE-2005-2773	NETWORK	LOW	NONE	PARTIAL
CVE-2006-1547	NETWORK	LOW	NONE	NONE
CVE-2006-2492	NETWORK	HIGH	NONE	COMPLETE
CVE-2007-3010	NETWORK	LOW	NONE	COMPLETE
CVE-2007-5659	NETWORK	MEDIUM	NONE	COMPLETE
CVE-2008-0655	NETWORK	MEDIUM	NONE	COMPLETE
CVE-2008-2992	NETWORK	MEDIUM	NONE	COMPLETE
CVE-2008-3431	LOCAL	LOW	NONE	COMPLETE
CVE-2009-0557	NETWORK	MEDIUM	NONE	COMPLETE
CVE-2009-0563	NETWORK	MEDIUM	NONE	COMPLETE

Figure 4. Example of CSV File with CVE Entries.

After all the data has been collected and stored into a CSV file, it is now time to extract all the KEV CVEs from the NVD CSV and remove all CVSSv3 CVEs from the NVD CSV and KEV CSV. The first portion of simplifying this data depends on using the FIND function within Excel to search each KEV cveID within the NVD CSV and removing the corresponding row, this will reduce our NVD CVEs from 218168 to 217154. Next is to remove all CVEs that only have CVSSv3 from each CSV. This solution has already been coded back within Figure 2, inside the else statement there is a print statement that can be used to weed out which CVEs do not have a CVSSv2. The code is simply printing out the cveIDs for all CVEs that did not key a CVSSv2 and with this data the FIND function within Excel can be used once again to search each CVE and remove them from each document. This process removes 126 CVEs from the KEV CSV, and it also removes 44605 CVEs from the NVD CSV, leaving a total of 870 KEV CVEs and 172549 NVD CVEs.

This research project will focus on three different approaches to analyzing data and calculating results. The first approach will be for the cybersecurity professional to randomly pick a certain number of CVEs to complete each day and then calculate how many CVEs they correctly patched that would end up being exploited and how many that

would end up wasting their time on. The second approach would be to focus on the CVSSv2 scores and create calculations on whether or not the CVE would end up being exploited based on how high the CVSSv2 baseScore is. For example, how many potential KEV CVEs would the cybersecurity professional correctly patch if only working on CVEs with a CVSSv2 baseScore of 10, 9 and above, 8 and above, 7 and above, etc.? From there we can calculate how many CVEs the cybersecurity professional would correctly patch that would end up being exploited and how many they wasted their time patching. Finally, the third approach to this research will be to use RapidMiner Studio's Auto Modeling and feed it sample data using the NVD CSV and the KEV CSV. A third Python file must be created that will take the NVD CSV as the input and randomly pick a certain exponential of the total KEV CVEs from the CSV and the random amount into a new CSV (Figure 5). From here the KEV CSV needs to be manually pasted into the new CSV with the randomly selected CVEs and a new column needs to be made to differentiate KEV from NOT KEV. The sample data is now completed and can be fed into RapidMiner Studio's Auto Model program for prediction using the new column labeling each as KEV or NOT KEV as the prediction factor. The system then runs and displays nine various models that were used to predict the data. This research will focus on the output that comes from the accuracy, so the printed data needs to be changed from classification error to accuracy.

```python
import csv
import random

# Input CSV file (change this to your file path)
input_csv_file = 'input.csv'

# Output CSV file (change this to your file path)
output_csv_file = 'output.csv'

# Number of rows to select (870 in this case)
num_rows_to_select = 1740

# Read the input CSV file and store rows (excluding the header) in a list
selected_rows = []

with open(input_csv_file, 'r', newline='') as csvfile:
    reader = csv.reader(csvfile)
    header = next(reader)  # Store the header row
    for row in reader:
        selected_rows.append(row)

# Ensure there are enough rows to select
if len(selected_rows) <= num_rows_to_select:
    print("Not enough rows in the input CSV to select from.")
else:
    # Shuffle the rows randomly
    random.shuffle(selected_rows)

    # Open the output CSV file for appending
    with open(output_csv_file, 'a', newline='') as csvfile:
        writer = csv.writer(csvfile)

        # Write the header to the output file (if it doesn't already exist)
        if csvfile.tell() == 0:
            writer.writerow(header)

        # Write the selected rows to the output file
        for row in selected_rows[:num_rows_to_select]:
            writer.writerow(row)

    print(f"{num_rows_to_select} randomly selected rows have been appended to {output_csv_file}.")
```

Figure 5. Example of Python File for Randomly Selecting NVD CVEs.

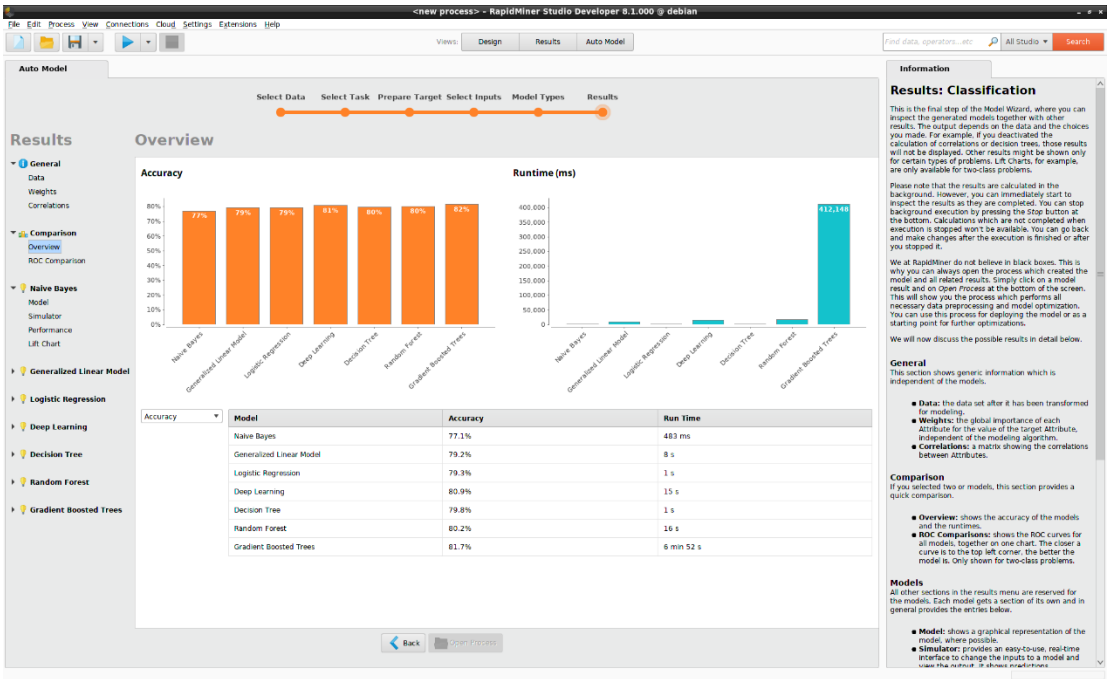

Figure 6. Example of Output from RapidMiner Studio's Auto Model

CHAPTER 4: OUTLINE OF COMPLETED THESIS

Here we have Johnny, he is a cybersecurity professional for a mom-and-pop shop whose job is to work from 9 am to 5 pm every day and patch incoming CVEs each day. However, during 2022 there was a total of 25101 CVEs published leaving Johnny with 69 CVEs every workday and he was not able to keep up with every incoming CVE and fell behind because asking someone to complete 12 CVEs per hour is inconceivable. With the following calculations, we hope to make Johnny's life and many others like him easier and show what data should be patched and what data should be skipped over, so cybersecurity professionals with too much on their plate do not have to stress about having to patch all 69 daily CVEs. For this solution, we are presenting Johnny with three different approaches we mentioned earlier. Before we can begin calculating the approaches, we must set some base values that are important for all approaches. Here we will estimate that Johnny can patch eight CVEs each workday resulting in 2080 a year because there are 260 workdays in a year, and then we will take the 172549 NVD CVEs from the NVD CSV (Figure 7).

$$8 * 260 = 2080$$

$$172549 \text{ Total CVEs}$$

Figure 7. Constant Values for Calculations.

The first approach is the previously discussed random approach where we will use the idea that a cybersecurity professional can complete 8 CVEs a day, allowing one hour for each CVE to be patched. With this approach, during 2022, Johnny would have randomly picked 8 out of the 69 CVEs he received each day to patch leading to 2080 a

year. To calculate the correct number of tickets he patched that would be added to the KEV database we must divide our 870 KEV CVEs by the total 172549 NVD CVEs and multiply it by 100 to get the percentage of CVEs that he would patch that would be exploited. We can now multiply that percentage by our 2080 CVEs completed each year, and we will end up with 10.5 or 11 (Figure 8). This concludes that of the 2,080 Johnny could complete each year, if Johnny randomly selected 8 CVEs to fix each day, out of the whole year he would correctly guess 11 CVEs that would be exploited and added to the KEV, but he would waste his time patching the other 2069 CVEs that would not be exploited. There must be a better way to predict which CVEs need to be patched and which ones can be ignored, so we move onto the second approach.

$$870 \div 172549 * 100 = 0.5042\%$$

$$2080 * 0.5042\% = 10.5 \text{ or } 11$$

$$2080 - 11 = 2069$$

Figure 8. Calculations for the First Approach.

The second approach (Figure 9) and probably the most logical will be executed by focusing on the CVSSv2 baseScore values based on their specific values because logically the higher the baseScore of the CVE, the more likely it is to be exploited. However, we can see that that is not always the case because the baseScore in the KEV database ranges from 1.9 to 10. With this approach, we will say that Johnny will only be patching CVEs with a baseScore of 10, 9 and above, 8 and above, and 7 and above. Starting with all KEV CVEs with exactly 10 baseScore equals 128 CVEs, 9 and above equals 293 CVEs, 8 and above equals 299 CVEs, and 7 and above equals 563 CVEs.

With these values, we can now calculate the percentage of CVEs that he would patch that would be exploited. To calculate the number of CVEs we need to divide by to get the correctly predicted percentage we must look at the NVD CSV file and sort by each tier of baseScore being used in the calculation. With this, we find there are 8043 CVEs with a baseScore or 10, 18956 CVEs with a baseScore of 9 and above, 19847 CVEs with a baseScore of 8 and above, and 53136 CVEs with a baseScore of 7 and above. Once all of the numbers have been pulled for the calculations you can now divide said numbers by the previously stated KEV CVE numbers for each baseScore tier and you will get the percentage of patched CVEs that would end up becoming a KEV. From there you will multiply the given percentage by 2080 and end up with the total number of CVEs that would correctly patch if he only patched CVEs with that specific CVSSv2 baseScore. From there you subtract that number from the 2080 total completed each year and you can see that as the sample size grew the percentage of correctly predicted CVEs falls. As expected, this method is much more efficient than randomly selecting CVEs to complete each day, if you were to replicate these same results with the random method you would need up to 4 employees all doing the random method when you could just use a singular employee using this one.

KEV 10	8043	128	1.59%	33	2047
KEV 9+	18956	293	1.55%	32	2048
KEV 8+	19847	299	1.51%	31	2049
KEV 7+	53136	563	1.06%	22	2058

Figure 9. Calculations for the Second Approach.

The third and final approach is to use machine learning where we fed data into RapidMiner's Auto Model and recorded the accuracy of data based on a randomly selected one-to-one ratio of NVD CVEs to KEV CVEs, as well as two-to-one, three-to-one, and four-to-one. The data was then run through RapidMiner's machine learning program utilizing the predict function focusing on the row that defined whether the CVE was KEV or NOT KEV, and then different percentages of accuracy were printed out based on the nine different models they drew from. From the one-to-one sample data of 870 NVD CVEs to 870 KEV CVEs the highest received accuracy was 71.6% from the Support Vector Machine model (Figure 10). We then had to execute the two-to-one sample data of 1740 NVD CVEs to 870 KEV CVEs, the highest received accuracy was 74.7% from the Random Forest model (Figure 11). Then, from the three-to-one sample data of 2610 NVD CVEs to 870 KEV CVEs the highest received accuracy was 77.0% from the Gradient Boosted Trees model (Figure 12). Finally, in the four-to-one sample data of 3480 NVD CVEs to 870 KEV CVEs the highest received accuracy was 80.0% from which there was a tie between Naïve Bayes, Logistic Regression, Fast Large Margin, and Decision Tree (Figure 13).

Model	Accuracy
Naive Bayes	68.9%
Generalized Linear Model	71.1%
Logistic Regression	67.0%
Fast Large Margin	65.6%
Deep Learning	68.6%
Decision Tree	65.2%
Random Forest	70.0%
Gradient Boosted Trees	68.9%
Support Vector Machine	71.6%

Figure 10. Calculations for the Machine Learning One-to-one Ratio.

Model	Accuracy
Naive Bayes	72.7%
Generalized Linear Model	73.5%
Logistic Regression	70.6%
Fast Large Margin	72.7%
Deep Learning	73.1%
Decision Tree	66.6%
Random Forest	74.7%
Gradient Boosted Trees	72.3%
Support Vector Machine	66.6%

Figure 11. Calculations for the Machine Learning Two-to-one Ratio.

Model	Accuracy
Naive Bayes	75.0%
Generalized Linear Model	76.8%
Logistic Regression	67.1%
Fast Large Margin	74.2%
Deep Learning	74.3%
Decision Tree	74.8%
Random Forest	74.8%
Gradient Boosted Trees	77.0%
Support Vector Machine	75.0%

Figure 12. Calculations for the Machine Learning Three-to-one Ratio.

Model	Accuracy
Naive Bayes	80.0%
Generalized Linear Model	79.3%
Logistic Regression	80.0%
Fast Large Margin	80.0%
Deep Learning	79.2%
Decision Tree	80.0%
Random Forest	78.3%
Gradient Boosted Trees	79.2%
Support Vector Machine	79.9%

Figure 13. Calculations for the Machine Learning Four-to-one Ratio.

As you can see, with each increase in ratio the percentage of accuracy increases, the increase in accuracy likely spans from having more data to train the machine learning algorithm allowing more information for it to compare each CVE. However, eventually the accuracy would start to decline due to the sheer amount of data if we continued to increase the ratio. With more data comes more outliers and anomalies that could alter the

predictions that the machine makes, it is important to make sure your data is high quality or the increase in data might affect the outcome of your accuracy.

Now that we have gathered the percentages for each ratio, we can now begin our calculations, we must create tables for each highest percentage and cross-multiple labeling one side with the total KEV CVEs and label the other side with accuracy percentage and the difference of one minus the percentage to get the classification error to calculate how many CVEs this approach will correctly predict and how many it will miss. Now we can cross multiply across the table, first, we take the accuracy percentage and multiply it by 870, this will give us the total number of correct CVSSv2 CVEs we would correctly patch over the span of twenty years because our data ranges from 2003 to 2023. From there you must divide the total number by 20 to get the amount you would correctly predict in a singular year; this calculation would not be accurate for each year because each year has a different number of CVEs but using this number as an example helps us get an explanation of what the result could be. Now you can take the classification error and multiply it by the total KEV CVEs and divide by 20 and we get the total number of CVEs the machine learning program would miss each year (Figure 14). You can see that with this approach we get nearly the same or less than the secondary approach however, this approach is preferable because it predicts CVEs based on many different factors rather than just specific CVE baseScores and it would be able to detect CVEs that would go undetected using the secondary approach. There is also the possibility of figuring out how to increase the accuracy of the machine learning data so that you can predict more than just 31 to 35 CVEs each year. With this approach, like the second one, you would also need up to 4 employees using the random method to match the same amount of accuracy.

One-to-One	71.6%	28.4%
870	623	247
Exploited CVEs Per Year		31
Missed CVEs Per Year		12

Two-to-One	74.6%	25.4%
870	649	221
Exploited CVEs Per Year		32
Missed CVEs Per Year		11

Three-to-One	77.0%	23.0%
870	670	200
Exploited CVEs Per Year		33
Missed CVEs Per Year		10

Four-to-One	80.0%	20.0%
870	696	174
Exploited CVEs Per Year		35
Missed CVEs Per Year		9

Figure 14. Calculations for the Third Approach.

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

Successfully predicting all possible CVEs that could be exploited and classified as a KEV is challenging, but not impossible. By using any of the explained approaches in this research it is possible to predict some KEV CVEs and reduce the impact of cybersecurity attacks. There may be many better ways to predict possible KEV CVEs or a way to perfect the approaches used in this research. We must work to perfect these approaches to assist cybersecurity professionals in reducing the number of CVEs they must complete and assist small businesses in being able to keep up with the massive number of incoming CVEs. It is crucial that these approaches are further looked at and perfected to assist cybersecurity professionals in reducing the amount of potential data breaches.

Future work consists of deeper analysis of machine learning, due to lack of time this research was unable to go into complete depth about what the machine learning program uses to predict future CVEs and how it can be used to give cybersecurity professionals the exact result of which CVE would be considered a predicted KEV CVE, this research merely states the accuracy of the machine learning predictions. Another area that could have been explored was further perfecting the accuracy of the machine learning program. I would have liked to maybe explore more approaches that could result in greater accuracy as well.

REFERENCES

Chang, Y.-Y., Zavarsky, P., Ruhl, R., & Lindskog, D. (2011). *Trend Analysis of the CVE for Software Vulnerability Management*. ResearchGate. https://www.researchgate.net/publication/220876036_Trend_Analysis_of_the_CVE_for_Software_Vulnerability_Management

Cybersecurity and Infrastructure Security Agency. (2023). *Known Exploited Vulnerabilities Catalog*. CISA. https://www.cisa.gov/known-exploited-vulnerabilities-catalog

Marshall, K. (2023, February 14). *How does the size of the training data affect the accuracy?*. Deepchecks. https://deepchecks.com/question/how-does-the-size-of-the-training-data-affect-the-accuracy/#:~:text=Furthermore%2C%20the%20training%20model%20accuracy,likelihood%20of%20overfitting%20the%20dataNational Institute of Standards and Technology. (2023). *National Vulnerability Database*. NVD. https://nvd.nist.gov/vuln/data-feeds

Targett, E. (2023, May 24). *Analysis of 26,000+ cves in 2022 shows shocking rise in...* The Stack. https://www.thestack.technology/analysis-of-cves-in-2022-software-vulnerabilities-cwes-most-dangerous/#:~:text=A%20record%2026%2C448%20software%20security,of%20publicly%20disclosed%20cybersecurity%20vulnerabilities