Automating cyber security advisories

Supervised machine learning for automated decision making

Stefan Weegink

Delft University of Technology

TU Delft

Automating cyber security advisories

Supervised machine learning for automated decision making

by

Stefan Weegink

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday September 27, 2022 at 10 AM.

Student number:	4650956	
Project duration:	November, 2021 – September, 2022	
Thesis committee:	Dr. A. Zarras,	TU Delft, supervisor
	Ir. S. Verwer,	TU Delft
	Dr. ir. J. Pouwelse,	TU Delft

Abstract

There is an everlasting struggle for organisations to remediate vulnerabilities in IT systems before being the victim of an exploitation. Organisations try to reduce this struggle by turning to specialized cyber organisations, which use their expertise to recommend resolving a subset of vulnerabilities. Unfortunately, the process of recommending a selection of vulnerabilities is primarily done manually. Manual labour is time consuming and requires skilled personnel. Automating cyber advisories reduces both these problems.

We introduce ACSA, a process designed for the Automation of Cyber Security Advisories. ACSA creates a dataset that can be used by advisory publishers to automate their publications with minimal effort. The dataset contains around 90,000 advisories which are filtered by a machine learning model to the set published by the organisation. We applied the ACSA process and dataset to both the Dutch and Canadian NCSC and found that on average we can already automate the majority of advisories. This constitutes a significant workload reduction in comparison to the situation prior to the automation. Even better results are observed when looking at the performance of ACSA on specific vendors. For some vendors we are able to automate more than 90% of the advisories while creating minimal false positives.

Keywords: automated advisories, patch prioritization, vulnerability prioritisation, exploit prediction, keyword extraction, supervised learning

Preface

This thesis is the final product of almost a year of work. Due to the COVID-19 pandemic, a large portion of my research was done from home. However, with frequent meetings online we found a good solution to deal with the situation. In the end, I am pleased with the end product, which could not have been accomplished without the help of several others.

First of all, I would like to express my gratitude to my supervisor Apostolis Zarras.In our weekly meetings every Thursday morning, we discussed progress and received advice from Apostolis as well as other students. Similarly, I had weekly guidance sessions with Jeroen van der Ham, my supervisor from the Dutch NCSC. His knowledge on the topic was very helpful and he helped me get to know many new concepts which were incorporated into this thesis. He also put me in contact with the EPSS SIG group, which was very helpful in getting data for my research. Another supervisor from the NCSC was Renout Schoen. He helped me out with the more technical aspects and has been of great value to improve my machine learning skills. I could not have reached the same results without his advice. Finally, I would like to thank all my friends and family who have provided the necessary moral support during this thesis. Without all the fun conversations, lunch & coffee breaks and feedback I received from them, this project would have been much harder and less enjoyable.

Stefan Weegink
Delft, September 2022

Contents

1 Introduction

As the number of vulnerabilities disclosed each year increases, it has become impossible for organisations to resolve them all. In 2021, for the first time, the number of known vulnerabilities in a year surpassed 20,000 [40]. Resolving a vulnerability takes a substantial amount of time, whether it is from assessing which assets are vulnerable to deploying and testing patches. Given that organisations have a limited capacity, the strategy of remediating all vulnerabilities is no longer a viable option.

Instead, organisations now try to resolve only a selection of vulnerabilities by prioritizing them, using techniques such as the Common Vulnerability Scoring System (CVSS). CVSS computes the severity of each vulnerability as a function of its characteristics. An organisation could decide to address only those vulnerabilities with a critical severity.

However, researchers have argued that CVSS is not suitable for this application [1, 55, 56] and there are problems with CVSS regarding consistency, dispersion and redundancy [2, 10, 59]. Consequently, other techniques might be preferable for an organisation e.g. filtering based on risk. The risk of a vulnerability depends on whether hackers are trying to abuse it. An attempted exploit is referred to as an exploit-in-the-wild.

Fortunately, exploits-in-the-wild only occur for a limited number of vulnerabilities. Householder et al. [25] find that of 75,000 vulnerabilities studied only 3,100 had public exploits in a span of six years and Jacobs et al. [28] observed 3.7% of all vulnerabilities were exploited. In the same paper, a machine learning model is presented that can predict with high accuracy which vulnerabilities are most likely to be exploited.

Risk-based filtering helps to reduce the workload, but clients often lack the expertise to apply the techniques correctly. This is where specialized cyber organisations like the Dutch NCSC come into play. They publish advisories on a selection of the vulnerabilities and, hence, we refer to these organisations as advisory publishers in this paper. These advisory publishers apply prioritizing techniques like CVSS and exploit-in-the-wild filtering to retrieve a subset that they deem relevant for their target audience.

This is a massive step forward, but efficiency could be increased even further since the current process of building advisories is mostly manual labour. Analysts collect and combine various sources to build new advisories. Next, they need to decide whether the built advisory should be published or not. Altogether, this is a time-consuming process and requires skilled personnel, which is scarce in today's market. Furthermore, in the entire process human can make mistakes, which could result in important vulnerabilities being missed.

1.1. Objective

In this thesis, we aim to build a dataset that serves as the basis to automate most of the process of publishing the advisories. The automation reduces the aforementioned problems regarding scarce personnel, human errors and delays. The automation will be achieved mostly by the creation of a dynamic dataset that organisations can download and with minimal effort apply to automate their advisories. In fact, an organisation only has to supply which vulnerabilities are published in each of their advisories. All other steps of the process are automated including collecting and combining vulnerabilities and evaluating whether they should be published. We demonstrate this by applying it to the advisories of both the Dutch and Canadian NCSC. With the focus of the research on the creation of the dataset, we arrive at the following research question:

RQ: How can we create a dataset that allows for automating cyber security advisories?

To arrive at an answer to this research question, we perform the following research tasks:

1. Propose a process for automating cyber security advisories
2. Creating a dataset using the process designed. This dataset should encapsulate all advisories which are likely to be published by an organisation.
3. Train a machine learning model to filter the dataset for both the Dutch and Canadian NCSC
4. Evaluate the performance of the model to verify the dataset was correctly created

The process we designed allows for both the creation of the dataset and applying it in an automated fashion with minimal effort. The process we propose is called ACSA, which stands for Automated Cyber Security Advisories. ACSA creates a large dataset, currently containing around 100.000 advisories, and is continuously updated as new vulnerabilities are released. ACSA allows advisory publishers to easily apply the dataset, they only need to collect a list of all advisories published and for each advisory which vulnerabilities it contains. Then, ACSA will automatically train a model to filter all the advisories in the dataset to the subset published by the organisation.

1.2. Document outline

This paper is organized as follows: first, chapter 2 provides the relevant background information required for this thesis. Next, we describe the related works in Chapter 3. In chapter 4 we present our proposed model and cover some aspects of how we will conduct the research. We collect all the data for the model for machine learning in chapter 5. Next, deployment and evaluation of the machine learning model are discussed in chapter 6. Chapter 7 discusses the results found in the previous chapters and the limitations of this research. Finally, we conclude this work by providing an answer to the research question in chapter 8. In the same chapter, we also provide ideas for future work.

2 Preliminaries

This chapter provides an introduction to the terminology and background information required for this thesis. First, a brief overview of machine learning techniques, classifiers and metrics is given. This will help to understand the design, training and evaluation of automated advisory models.

To build these models, data needs to be acquired from various datasets. In the data section, we cover which information each dataset contains and how it will be used in this research. After that, we will cover several cyber security standards that are mentioned and used in this paper. Finally, a list of vocabulary is presented to prevent any ambiguity of terms used.

2.1. Machine learning

Machine learning is a subfield of Artificial Intelligence (AI). AI is commonly defined as the capability of a machine to imitate human behaviour. This is used to perform complex tasks and solve problems in a way similar to how humans do. For example, machines can recognize objects present in images or understand text written. Machine learning is the research field that gives systems the ability to learn without explicitly being programmed. Machine learning uses samples of data and tries to create a model that approximates a certain task. In general, feeding more examples to the algorithm improves the training of the model and the model should become more accurate in performing its task. Using machine learning allows software applications to execute tasks without being explicitly programmed to do so. This property makes it perfect for automating tasks while being applicable in a wide spectrum of fields. Notable examples are cancer prognosis [34] and detecting credit card fraud [8]. Within the cyber domain, there are also numerous applications of machine learning like intrusion detection [12] and attack prediction [39].

Within machine learning, there are two main approaches for training a machine learning model: unsupervised and supervised. Unsupervised learning tries to predict a task without knowing the expected outcome also referred to as the label. It tries to predict this by trying to find a structure within the unlabelled data and, for example, giving outlier a different label. In contrast, supervised learning requires labelled data. Using the label, it tries to find the decisions that result in the model best predicting the label. The latter approach is the one that applies to this research, with the label indicating whether an advisory has been published or not.

For machine learning, there are several options for data encoding, classifiers and techniques to use. The different options regarding these topics are covered in the coming sections, whereas the machine learning metrics will be covered in chapter 4.

2.1.1. Data encoding

Machine learning models require a specific data format. Generally, models can apply numeric data but fail to process categorical data. Data encoding solves this problem by converting categorical data into numerical data.

There are several encoding methods, which give different numerical representations of the original data. The methods considered in this paper are label encoding, one-hot encoding and frequency encoding. They will be explained using the following data:

CVE ID	Exploit status (cvss:E)	CVSS Score
CVE-2016-5894	Unproven	5.1
CVE-2019-11682	Proof of Concept	9.8
CVE-2020-10238	Unproven	5.3
CVE-2021-24724	High	6.4
CVE-2021-40398	Proof of Concept	8.1

Table 2.1: Sample vulnerability data

Label encoding

Label Encoding is the technique of converting categorical variables into single numeric values. For each variable, the technique assigns a unique number to each distinct value found.

Applying this method to the Exploit status in the sample data yields the result shown in table 2.2. The exploit status has three unique values, which are mapped to the values one to three in order of appearance starting from the top.

CVE ID	Exploit status (cvss:E)	CVSS Score
CVE-2016-5894	1	5.1
CVE-2019-11682	2	9.8
CVE-2020-10238	1	5.3
CVE-2021-24724	3	6.4
CVE-2021-40398	2	8.1

Table 2.2: Label encoding

Unfortunately, the conversion to numeric values, if used in the wrong way, could reduce the performance of the machine learning model. The model could obtain a false notion of distance when it for example compares or sums up values, while the values are not comparable or have hierarchical values. In this way, the encoding undermines the sense of ordinality in the data.

To illustrate, consider the case in which the order of encoding is changed. With the new mapping, the value of Proof of Concept and Unproven are swapped. Where the original order mapped values in order of how likely there is an exploit available, this now is no longer the case. Similarly, summing the exploit status values for several CVEs provides less value to a machine learning model. This is due to the hierarchical order being incorrect after swapping the values. Where previously a higher value meant a higher chance of exploit, this is no longer the case as unproven has a higher value than Proof of Concept. So, when applying label encoding the mapping should be carefully done to achieve the optimal result.

One-hot encoding

This method splits a single column with categorical data into multiple binary columns for each unique value present in the original column. In the sample data, the exploit status contains three unique values, resulting in three new columns being created. Each column is filled with ones for the records in which the unique value appears and zeros in the remaining cells:
The main issue with this method is the number of binary columns that may be created. A very large number of columns could result in performance and storage issues.

CVE ID	Status:Unproven	Status:PoC	Status:High	CVSS Score
CVE-2016-589	1	0	0	5.1
CVE-2019-11682	0	1	0	9.8
CVE-2020-10238	1	0	0	5.3
CVE-2021-24724	0	0	1	6.4
CVE-2021-40398	0	1	0	8.1

Table 2.3: One-hot encoding

Frequency encoding

Frequency encoding replaces each unique value in a categorical variable with a numeric value representing its relative frequency in the data set. It counts the number of occurrences of a value within a column and divides this by the total number of values within that column. The downside of this approach is that the categorical value is lost. Hence, frequency encoding is often applied in combination with one-hot encoding to ensure that the value itself is not lost.

CVE ID	Exploit status	CVSS Score
CVE-2016-5894	2/5	5.1
CVE-2019-11682	2/5	9.8
CVE-2020-10238	2/5	5.3
CVE-2021-24724	1/5	6.4
CVE-2021-40398	2/5	8.1

Table 2.4: Frequency encoding

2.2. Machine learning training techniques

Machine learning training is critical for the performance of a model. Within the training phase, several methods are available. First, we cover cross-validation and hyper-parameter tuning, which are techniques for the initial training of a model. Then, we cover feedback loops, which are used for the retraining of the model.

2.2.1. Cross-validation

Cross-validation is a method used for more consistent evaluation of machine learning models. In machine learning, the data is split into a training and a testing dataset. The split is often picked at random and hence, the performance of the model can deviate between consecutive runs. Furthermore, selecting just a single split might give an inaccurate indication of the performance of the model. The single split might have a lower or higher performance in comparison to if the data had been split in a different way.

To resolve this issue, cross-validation resamples different portions of the data to test and train a model on different iterations. For our research, we use the stratified K-fold cross-validation. With stratified K-fold the data is split into K folds, where K is an integer often chosen as 10. Every fold is split into a training and test dataset, which are used to train and evaluate a classifier. Then we average the scores from the different folds to obtain the final score. This reduces the deviation between consecutive runs which allows for a better comparison of performance.

2.2.2. Hyper-parameter tuning

Machine learning models have parameters for fine-tuning. These parameters are called hyper-parameters. We use the default parameters in the early stages of the model and use automatic hyper-tuning to find the best parameters for the final version. Automated hyper-tuning can be done using various algorithms but most notable are grid search and Bayesian optimisation. Grid search is a brute force solution whereas Bayesian optimisation is a more efficient solution.

2.2.3. Feedback loops

A feedback loop, which is also known as closed-loop learning, is the process of leveraging the output from a machine learning model to retrain and improve the model. The predicted label that is outputted by the system is compared to the actual label and the result of this comparison is fed back to the machine learning model. This feedback allows the model to learn from classifying mistakes and improve over time. For example, our model outputs that a certain advisory should be published whereas analysts decide the opposite. Providing this as feedback to the model and retraining the model, the model in the future might be able to correctly classify similar cases. Without the feedback loop, the model would make the same mistake. The feedback loop allows for the incorporation of new advisories into the training set, making the model adapt to evolving data. Evolving data can be the result of policy changes or a shift in the cyber threat landscape.

2.3. Classifiers

Machine learning models use classifiers to predict an outcome, which in our case is whether an advisory will be published or not. There exist numerous classifiers which can be used for the process. In this section, we elaborate on the classifiers chosen. For the classical classifiers, we will use logistic regression, decision trees and random forest. As for the more advanced algorithm we use XGBoost.

2.3.1. Logistic regression

Logistic regression uses a mathematical function for classifying objects. It is similar to linear regression but rather than using a linear function it uses a logistic function. The logistic function is as follows:

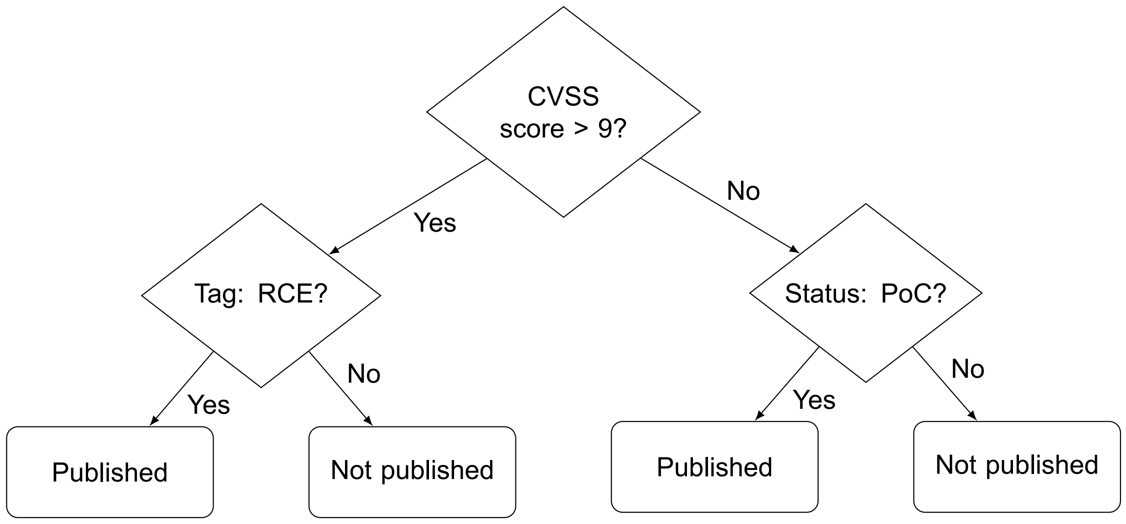
$$\begin{gathered} logistic(t(x)) = \frac{1}{1 + \exp(-t(x))} \\ t(x) = \beta_0 + \beta_1 * x_1 + \dots + \beta_n * x_n = \beta_0 + \sum_{i=1}^{n} \beta_i * x_i \end{gathered} \tag{2.1}$$

This function applies the logistic function on the record x, which contains n features. t(x) multiplies the weight (β_i) of features with the respective value (x_i) for each feature i.

The β weights are computed by the machine learning model to predict the labels to the best of its ability. The value found using the logistic functions needs to be mapped to a binary classification. This is done by applying a threshold, which by default is 0.5. Anything above this value is an advisory which we label as published whereas below is not published. In our research, we select the threshold that results in the best performance for the metrics chosen. With its binary classification and usage of the logistic function, the logistic regression classifier is highly efficient and usable for this research.

2.3.2. Decision tree

Decision tree classifiers, like their name implies, use trees for labelling. The decision tree depicted below takes each generated advisory and maps it to one of the leave nodes. This is done by starting at the root node and depending on the answer for the decision at this node it takes either the left or right child. This process is repeated until there are no more child nodes and the advisory receives the label of the leaf. For example, using the tree below and an advisory with a CVSS score of 10 and the tag remote code execution (RCE) will be labelled as published. The tree gives clear insight into how the labelling works. This allows organisations to easily evaluate their current publication process, which is a pro of using decision trees.



2.3.3. Random forest

In the random forest classifier, a large number of decision trees operate together. Each individual tree in the forest outputs its prediction and the class with the majority determines the final label.

For instance, consider the scenario in which we build a random forest with six decision trees. Four decision trees output the 'published' label while the other two output 'not published'. As the published label has the majority that will be the predicted label for the advisory.

Random forest should in theory perform better than a decision tree as its decision boundary becomes more accurate as more trees are added to the model. This potential increase of performance comes at the cost of a longer run-time.

2.3.4. XGBoost

Extreme Gradient Boosting (XGBoost) is based on several machine learning concepts like boosting. Boosting is a machine learning technique used to increase the performance of models. This is done by sequentially adding models to reduce the errors until no further improvements can be made. Random forest is an example of a boosting algorithm for the decision tree. It adds several decision trees to produce a new outcome with ideally less errors.

Gradient boosting is a more specific version of boosting in which new models are created that predict the errors of prior models and are combined to make the final prediction. The name gradient boosting comes from the models that use a gradient descent algorithm to minimize the loss when adding new models. Similar to random forest, the gradient boosting in XGBoost works with decision trees. The classifier has several parameters which determine how the boosting takes place. There are parameters that impact the number of boosting rounds the model it performs but also how different the trees are between consecutive boosting rounds.

In each boosting round, a decision tree is created which maps a record from the root node to a leaf node. The leaf node contains the raw score which is added for each tree created in the boosting process. A negative score for a leaf lowers the chance of the record being predicted with the True label while a positive score increases it. After summing the leaf values for all trees, we convert the raw score to a probabilistic score using eq. (6.1). Using a chosen threshold, the values above it are mapped to True label and below to not false label which in our case respectively are whether an advisory is published or not.

$$Probabilistic\_score = \frac{1}{1 + \exp(-1 * leaf\_value)} \tag{2.2}$$

Due to the gradient boosting, XGBoost has a very high performance and with its implementation being very efficient, its execution time is fast as well. Hence, many researchers are using XGBoost as their classifier of choice. XGBoost has two additional benefits. It is very suitable for tabular data (as used in this thesis) and it is widely used, which makes it relatively easy to compare results to other research.

2.4. Data

To build a dataset that contains all possible advisories, several different data sources are needed. These sources contain different information about vulnerabilities e.g. one source describes the products affected by a vulnerability whereas another shows whether it has been exploited in the wild. Furthermore, we also need to evaluate our dataset which is done using the Dutch and Canadian NCSC. We discuss the background of both organisations and any relevant information for how we extract the information later on.

2.4.1. CVE

MITRE's Common Vulnerabilities and Exposures (CVE) is a project with the goal to "identify, define, and catalog publicly disclosed cybersecurity vulnerabilities"[37]. Since 1999, MITRE has been working on that goal by creating a dictionary that contains all the identified vulnerabilities in software products.

Identified vulnerabilities will be assigned an unique "CVE ID", which is used by interested parties to acquire more information about certain vulnerabilities. In recent years, the CVE ID has become the industry standard. Vendors use it for publishing new vulnerabilities and researchers use it in papers.

CVE IDs are written in a specific format (CVE-YEAR-ID) which makes it easy to extract these IDs from text using a simple regular expression.

2.4.2. NVD

The National Vulnerability Database (NVD) was founded in 2000 by the US National Institute of Standards and Technology (NIST). It is a repository that contains data useful for vulnerability management. NVD focuses on providing all relevant information related to the vulnerability. This information allows an organisation to understand whether and how a vulnerability affects them allowing them to decide whether or not the vulnerability should be remediated. All the data in NVD is in a standard format such that organisations can automate the process of vulnerability management.

The database builds upon the CVE records, which it is synchronized with. NVD expands each vulnerability identified via the CVE ID with analysis performed by their own staff. This analysis concerns aggregating data points from the description and references supplied by CVE but also any additional data that can be found is used. This additional data can for instance originate from the vendor who fabricates the product that is affected by the vulnerability. The analysis produces information on the impact (CVSS), vulnerability types (CWE), affected products (CPE) and other metadata. As additional information becomes available, the CVSS scores, CWEs, and CPEs are subject to change. We cover these data types in-depth in the coming sections.

2.4.3. CVSS

The Common Vulnerability Scoring System (CVSS) is a mechanism that captures characteristics and severity of software vulnerabilities. Based on the characteristic and weighs assigned by CVSS, a numerical score reflecting the severity of a vulnerability is produced. The numerical score can be translated into a qualitative representation e.g. low, medium, high, critical for version 3.1 of CVSS. This qualitative representations can help organizations properly assess and prioritize their vulnerability management processes.

The characteristics CVSS uses are grouped into three metric groups: base, temporal and environmental. The base characteristics are constant over time and constant across different user environments. The temporal and environmental metrics on the other hand, change overtime or have a different user environment, respectively.

The base metrics are used to produce a score ranging from 0 to 10 using the weights assigned by CVSS to specific metric values. The produced score can then be changed by the temporal and environmental metrics. The environmental should be assessed by analysts for each individual organisation. This clearly conflicts with our aim to create a generic and non-labour-intensive solution for automating cyber advisories. Hence, environmental metrics are the only CVSS group that will not be used in the dataset. We collect the base metrics from NVD and temporal metrics from IBM X-Force as they are not included in NVD.

A CVSS score is also represented as a vector string, a compressed textual representation of the metrics and their values used to produce the score. For example a base vector string is ”CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H”. In table 2.5 and table 2.6, we explain the different metrics for respectively the base and temporal vectors. The possible values for each metric are visible in (Appendix komt eraan) The individual metrics can be used for machine learning as done in [29], which uses one-hot encoding on the metric values.

Metric	**Description**
Attack Vector (AV)	Reflects the context by which vulnerability exploitation is possible
Attack Complexity (AC)	Conditions beyond the attacker’s control that must exist in order to exploit the vulnerability
Privileges Required (PR)	Level of privileges an attacker must possess before successfully exploiting the vulnerability
User Interaction (UI)	Requirement for a human user, other than the attacker, to participate in the successful compromise
Scope (S)	Whether a vulnerability in one vulnerable component impacts resources in components beyond its security scope
Confidentiality (C)	Impact to the confidentiality of the information resources
Integrity (I)	Extent to which an attacker can modify acquired resourced
Availability (A)	Impact on access to resources and services

Table 2.5: Base metrics

Metric	**Description**
Exploit Code Maturity (E)	The likelihood of the vulnerability being attacked (Exploit-in-the-wild, PoC, etc.)
Remediation Level (RL)	Level to which a vulnerability can be remediated e.g. fix is available
Report Confidence (RC)	The degree of confidence in the existence of the vulnerability and the credibility of the known technical details

Table 2.6: Temporal metrics

Versions
The latest version of CVSS is 3.1, which is the version that we used so far and will be primarily used in this paper. It was released in 2019 and since then has become the standard. However, there are older versions of the model that are also relevant to this research:

- 3.0, which was released in 2015
- 2.0, which was released in 2007

The newer versions of the model try to tackle issues that will be discussed in the next chapter. Nevertheless, all versions work similarly as the newer versions expand upon the older version. Furthermore, sources like NVD disclose information in multiple CVSS versions. Within NVD all vulnerabilities since 2007 have the CVSSv2 vector. Vulnerabilities found after CVSSv3 initial release also contain the CVSS vector for the latest 3.x version that was available at the time of disclosing the vulnerability. So, any new vulnerability at the moment receives both a CVSSv2.0 and CVSSv3.1 vector. We will use both these vectors as both are used for vulnerabilities.

As both versions are versions, it is essential to understand the few differences between them. The newer versions make several changes to the base group while the temporal group stays the same. The biggest change is the introduction of the metrics User Interaction (UI) and Privileges Required (PR). Another change was the addition of the Physical (P) value to the Attack Vector (AV) metric. Finally, they changed the naming of the values of the Confidentiality, Integrity and availability. The metrics were originally named None, Partial and Complete, which was renamed to None, Low and High. The last change made is qualitative scoring mappings where the high severity was split in to high and critical severity. The rest of the qualitative mapping remained unchanged, resulting in the following qualitative mapping for CVSSv3.1:

- Low, 0.0 – 3.9
- Medium, 4.0 – 6.9
- High, 7.0 - 9.0
- Critical, 9.0 - 10.0

2.4.4. CWE

The Common Weakness Enumeration (CWE) is a category system for both software and hardware weaknesses. The category system is community-developed and helps understanding the nature of flaws. CWE has over 600 categories, including specific types of buffer overflows or cross-site scripting. CWE uses a hierarchical structure, which differs depending on the view used. Views create different groupings of weakness for different usage scenarios like research or product development. In this paper, we will use the research concepts view as it is the best fitting and standard for any scientific research. The other views are used for efforts not related to research but for example focus on hardware development.

In the research concepts view, there are 927 weaknesses. These are mapped in a tree structure that starts with generic descriptions and becomes more specific with each level. For example, a leaf node with a specific weakness is CWE-121 Stack-based buffer overflow. This weakness is has the following hierarchical structure:

1. CWE-664: Improper Control of a Resource Through its Lifetime
2. CWE-118: Incorrect Access of Indexable Resource ('Range Error')
3. CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer
4. CWE-787: Out-of-bounds Write
5. CWE-121: Stack-based buffer overflow

This is a simplified version of the actual hierarchical structure as weaknesses can be children of multiple weaknesses. In fact, CWE-121 is a child of not only CWE-787 but also CWE-788. The entire structure is documented by MITRE and NVD assigns CWE identifiers to vulnerabilities. NVD only assigns the CWE identifier with the lowest level. Using the trees, we can find all the other CWE identifiers that are also associate to that vulnerability.

2.4.5. CPE

The Common Product Enumeration (CPE) is an initiative similar to CWE that is maintained by NIST. Where CWE provides a structured system for categorizing vulnerabilities, CPE structures the name of IT products. NIST integrates the CPE data into the NVD dataset for easy application. The NVD dataset currently uses version 2.3 of CPE dictionary, which is the latest version at the moment.

CPE consists of a generic syntax that is based upon the generic syntax for Uniform Resource Identifiers like URLs. CPE uses a syntax in which different types of information are split using ':'. The full syntax for version 2.3 of CPE is as follows:

$$cpe :< cpe\_version >:< part >:< vendor >:< product >:< version >:< update >:< edition >$$
$$:< language >:< sw\_edition >:< target\_sw >:< target\_hw >:< other >>$$

The variables in this format are defined as shown in table 2.7. The values for each variable are restricted to a set of characters. Even more restricted are the values for vendor and products which are drawn from an agreed upon list of valid-values. This is to ensure consistency between several CPE strings. Furthermore, most often the values for update, built and language are left empty which in CPE equates to the * operator. This means when search

Variable	Description
cpe_version	The version of the CPE definition. The latest CPE definition version is 2.3
part	the type of product; o (operating system), a (applications) or h (hardware)
vendor	identifies organisation that manufactured the product
product	the name of the system, package or component
version	the version of the system, package or component
update	the minor version of system, package or component
built	the build of the system, package or component
language	language tag as defined by IETF RFC 4646 like en-us

Table 2.7: CPE variables

Using the syntax and variables defined above, we build several CPE strings and explain their definition if present in NVD dataset:

- cpe:2.3:o:microsoft:windows_7:-:*:*:*:*:*:*:*, any version of the windows 7 operating system from Microsoft is affected by the vulnerability.
- cpe:2.3:a:ntp:ntp:*:*:*:*:*:*:*:*, any version of the NTP application from NTP is affected by the vulnerability
- cpe:2.3:a:microsoft:internet_explorer:8.0.6001:beta:*:*:*:*:*:*, the beta of version 8.0.6001 of internet explorer is affected by the vulnerability

The CPE strings can also be used in the NVD API to find all vulnerabilities which affected a certain product or vendor. The NVD API matches any CPE string which has the same exact value for a variable or contains the * operator in the search string.

2.4.6. Dutch NCSC

The Dutch National Cyber Security Centre (NCSC) tries to protect the Dutch digital infrastructure. This is vital as a cyber incident on a Dutch organisation can have a devastating impact, for instance no drinking water. The NCSC tries to prevent such incidents by providing security guidelines, publishing advisories and being a primary party in the National Detection Network (NDN).

The NDN is an intelligence pool where organisations work together to increase the Digital Security of the Netherlands. The NDN consists of governmental organisations but also companies critical to the Dutch infrastructure. Within the NDN, the organisation share threat intelligence such that cyber risks and incidents can be prevented or impact reduced. The Dutch NCSC shares general intelligence on active cyber threats, which are a potential risk to the organisations in the NDN.

Another task the NCSC performs is releasing cyber security advisories to inform organisations on which patches to apply to remediate vulnerabilities. Analysts monitor thousands of sources to identify patches which are relevant to Dutch organisations. They check if the vulnerabilities are relevant by, for instance, checking which products are affected by the vulnerability and whether these are used in Dutch organisations. Then the NCSC analyses the patch and collects extra information to compute the likelihood and damage scores. These score are mapped to a qualitative scale (Low, Medium, High) which is used to indicate the importance of an advisory to clients. The likelihood focuses on all CVSS base and temporal vector metrics except for the CIA triad. The confidentiality, integrity and availability are contained within the damage scores. Furthermore, the NCSC provides a description, characteristics, information on affected assets, CVE IDs and patches. All this information provides potential useful features for the models that will be trained as well as that the model can be used to compare the scores. After the advisory is created, it is checked by another analyst and again verified if the advisory should be published.

The Dutch NCSC does most of the process manually. Only the source collection and advisory publication is done automatically. The processing of the sources, building advisories and deciding whether the advisory should be published is manual labour. Using, the dataset produced in this paper the NCSC can easily automate all the steps of the process reducing workload.

2.4.7. Canadian NCSC

The Canadian Centre for Cyber Security (the Cyber Centre) is an institution similar to the Dutch NCSC as its name already hints on. The Canadian NCSC is the single unified source of expert advice, guidance, services and support on cyber security for Canadians [44]. The various informational sources the NCSC discloses on its website can be filtered using different target audiences. The available target audiences are individuals, small & medium businesses, large organisations and infrastructure, government institutions & academia. Whereas the individual filters on less technical articles with a focus on awareness, while academia also includes highly technical articles. These more technical articles explain how cyber security concepts like Identity and Access Management work.

2.5. Security standards

In recent years, the cyber security industry has released several standards which help our research as they standardize the way data is distributed, making the data collection easier. Likewise, we also these standards in our research such that our results can be easily applied and understood.

2.5.1. CSAF & CVRF

Oasis's common security advisory framework (CSAF) provides a template for cyber security advisories. The template describes which information should be included in an advisory and how it should be structured. CSAF is the renaming of an older project known as the Common Vulnerability Reporting Framework (CVRF). Since the renaming, several changes have been made to the framework with the main revision being the changing from XML to JSON as a format. Many of the other features remained the same across the different versions, resulting in both terms being interchangeably used. CVRF and CSAF still both rely upon the same document structure called Vulnerability Exploitability eXchange (VEX). Organisations incorporate the VEX document structure and can then publish advisories in either CSAF or CVRF. By now many large vendors like Microsoft and Cisco have adopted CSAF but also still publish in CVRF.

The standardizing of cyber security advisories is beneficial to this research. We are automating the publication of cyber security advisories for organisations like the Dutch NCSC. These organisations collect advisories from vendors and publish on a subset of the advisories. Prior to CSAF, the collection and processing of advisories in an automated manner was difficult. It required advanced techniques to extract the desired information from the unstructured resources. As CSAF is not adopted by all organisations, we use both the advanced techniques and CSAF to collect the information from vendor advisories.

2.5.2. VEX

The Vulnerability Exploitability eXchange (VEX) is a document structure that allows an instance to assert the status of a specific vulnerabilities in a particular product. VEX is supported within CSAF and CVRF to allow vendors to provide the status of the vulnerabilities that may affect a product. In this sense VEX provides a form of a security advisory that replaces the traditional unstructured advisories. In contrast to the traditional advisories are VEX documents machine readable. This allows for easy integration into vulnerability management software.

Within a VEX document there are three main sections: document, product tree and vulnerabilities. The document describes typical information for documents, such as timestamp, CSAF version and author. The product tree section is tree structure which includes all information on all products affected by the vulnerabilities. The Tree structure shows to which product group and vendor a specific product belongs. In the final leave node a product_id is specified which corresponds to a unique version of the product. The product_id is used within the vulnerabilities section to describe the vulnerability status for several products.

The vulnerabilities section enumerates various vulnerabilities. For each vulnerability the VEX document contains a title, description, CVE ID, release date, CVSS scores, threats and product statuses. The product status sections shows displays a categorical value for each product id. The categorical values available are *known_affected*, *known_not_affected*, *under_investigation* and *fixed*.

2.5.3. TLP

FIRST's Traffic Light Protocol (TLP) is a protocol designed to make sure sensitive information is only shared with the intended audience. TLP describes four colours to indicate the sharing boundaries that should be obeyed by the receiving parties:

- **Red**, indicates information that cannot be shared with parties outside of the specific exchange.
- **Amber**, indicates information that can be shared with members within the organisations taking part in the exchange of information
- **Green**, indicates information that can be shared with any peers and organisations within the same sector or community
- **White**, indicates any information that is not subject to restrictions

In this research we will only used TLP:white information, because access to any other TLP data was not granted by the NCSC. Furthermore, it ensures anyone can validate the research done. The usage of only TLP:white might impact the performance of our model. For example, consider the scenario in which Microsoft signals a vulnerability to NCSC with TLP:red. This might be the reason for NCSC to publish an advisory, which a machine learning model would not predict as published due to lacking information.

2.5.4. STIX

Structure Threat Information eXpression (STIX) is a language and serialization format used to exchange cyber threat intelligence (CTI). STIX aims to provide a standard for cyber threat information terms and their definitions. The terms we use from STIX are IOC, Threat Actor, TTP, COA and Exploit target, which are described in the next paragraph. STIX is also used in several related works such as [49], which provides a foundation for this research.

3 Related Work

In recent years much work has been done related to improving vulnerability remediation. Some of it is even directly related to automating cyber advisories. As mentioned in the introduction, automation of cyber advisories requires two components. On the one hand, we need to collect data to build the dataset containing all possible advisories. On the other hand, we need to filter the dataset to obtain the subset of published advisories. Hence, we need to acquire relevant knowledge on both topics.

The data collection is done by using APIs, datasets and unstructured sources like webpages. While the first two are relatively easy, the last poses a challenge to extract the useful information. The general research field that relates to the extraction of information from unstructured textual sources is natural language processing (NLP). Within NLP, there are various sub-fields like keyword extraction which are more relevant to the research conducted in this paper and which are also used in earlier attempts of automating cyber security advisories. Hence, we cover related work on the topic of keyword extraction.

The filtering of the advisories will be done by utilizing techniques studied in the vulnerability prioritization field. This is a very active field of research and contains many papers on CVSS but also other more advanced prioritization techniques. We cover step by step how the field of vulnerability prioritization has developed throughout the years in the coming section until we reach EPSS. EPSS is one of the most advanced vulnerability prioritization techniques and is an important source of information for the filtering performed in this paper.

After having discussed both these components, a brief study of other research that aims at improving vulnerability remediation is discussed. This research, which we refer to as patch management, might make certain steps in our model proposed in chapter 5 easier or even obsolete.

3.1. Vulnerability prioritization

Whereas automating advisories could be regarded as a quite novel field of research, vulnerability prioritization has been around for decades. Research in the field started with attack prediction as only much later the load of resolving all vulnerabilities became a problem. Attack prediction uses machine learning models to predict when and how attacks will take place [17, 47]. It can help with early detection of attacks [24] as the analysts are on high alert if an attack is anticipated.

Although its orientation is towards the detection of attacks, this field of research uses relevant machine learning techniques and data sources. For instance, Okutan et al. [47] show that social media, like Twitter, is a valuable source for features like how often an vulnerability is mentioned on the platform. Such features are used to develop models that outperform those which are today's industry standard for vulnerability prioritization. Before these advanced models are elaborated further, we first discuss these industry-standard models as they provide the foundation for the improved models.

CVSS, which was covered in detail in the previous chapter, is such an industry-standard model for ranking vulnerabilities [11, 28]. Despite being widely used for prioritizing vulnerabilities, CVSS is argued to be unsuitable for this purpose as CVSS's base score captures the severity of a vulnerability, not the risk. Consequently, it does not capture the probability that a vulnerability will be exploited. This limitation of CVSS was demonstrated by Allodi and Massacci [1], who found that using an older version of CVSS to prioritize vulnerabilities performs equally to picking vulnerabilities at random.

Although newer versions of CVSS have been released, the problem persists as is argued in [55]. Springer et al. argue ”that the current CVSS version is still inadequate for both its intended purpose and as a proxy for risk to a vulnerable system”. Furthermore, there are some deficiencies in the dispersion of CVSS scores [59], scores are not consistently calculated [2] and part of the CVSS metrics can be considered redundant [10].

With all this critique towards CVSS, researchers have been looking into models that have a better performance. Spring et al. [57] resolved part of the issues discussed in [55] by proposing a new model called SSVC. SSVC uses custom build decision trees that attempt to not just capture severity but also the overall risk. The model is not intended to predict future exploitation, it rather tries to acknowledge the current state of affairs [55]. Although the model seems an improvement in comparison to CVSS, it has not been empirically validated. Spring et al. validate by showing that different analysts retrieve the same outcome from the model. Despite its lacking validation, it still proposes interesting ways to capture risk in models which we see being used in the advanced vulnerability prioritization models.

Next to SSVC, numerous other models have been proposed for vulnerability prioritization. [11] is such a model, it trains a classifier that is able to tell whether and when a vulnerability will be exploited. Although, it achieves an accuracy of 89.8% such research is outdated. It uses vulnerabilities prior to 2007 and the dataset OSVDB, which has been terminated since 2016. Similar problems come to light in several other research papers, for example, the usage of Symantec's WINE or SYM in [3]. These datasets are no longer publicly available since Symantec was acquired by Broadcom. Nevertheless, those papers still have findings that can be used for further research.

Allodi et al. [3] find that vulnerabilities with a lower complexity or higher impact yield a higher chance of exploitation. This indicates that CVSS, which encapsulates the complexity and impact of a vulnerability to a certain degree might be useful to incorporate into our machine learning model. In addition, Allodi et al. [3] uses exploit in the wild data rather than proof of concept (POC). Some older research uses the latter as outcome metric for machine learning models [4, 11, 16]. Although POC data is easier to collect, it says little about exploitation in the wild [1]. More recent papers tend to use this exploit in the wild data [28, 53] as it shows whether exploitation of a vulnerability has occurred in the real world. Acquiring exploit in the wild data is a more cumbersome task and, hence, researchers utilize several non-obvious sources of threat intelligence, including Twitter [22, 53], the dark web [4, 22], hacker forums [22] but also private datasets of commercial organisations [28]. Despite using private datasets, the latter is the state-of-art model for vulnerability prioritization. This model will be used for filtering advisories but to grasp how it can be used for this purpose, one first needs to understand how the model works.

The Exploit Prediction Scoring System or abbreviated as EPSS was first introduced in [28]. EPSS is a model which predicts the likelihood that a vulnerability will be exploited in the wild. The model uses a diverse feature set comprised of exploit in the wild data, exploit code, CVE and NVD to build a model. The model has one of the largest exploit in the wild dataset available finding that 3.7% of all CVE's were exploited [29] while previous research finds only 1.4% [53]. Using this exploit in the wild data as outcome feature, the model outputs scores between 0 to 1 where a higher score indicates a larger chance of exploitation. The following diagram shows which 16 features are most important for predicting the score:

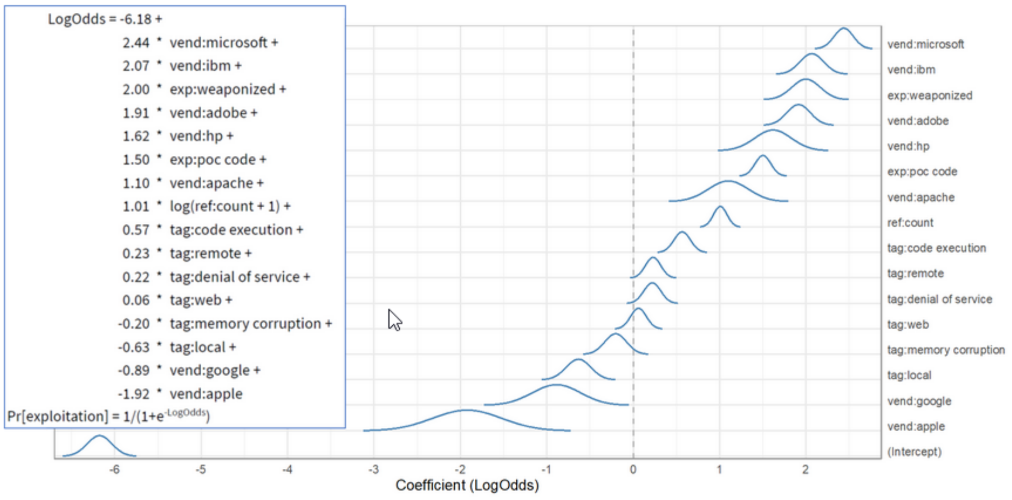


Figure 3.1: Most significant features in EPSS

Over the years, FIRST has further refined the EPSS model. In the first version [28], the relevance of the model was mostly shown and the potential of a very simple model. In the following papers, [27, 29] they expanded the model with new features or improved those already present. The features added or improved are interesting to our research like CVSS, number of references, tags and exploit status features. Hence, we investigate how they collect these features. Concerning the exploit status they have four features; exploit-in-the-wild, GitHub, Metasploit and ExploitDB. Exploit-in-the-wild is collected using a private dataset and can therefore not be duplicated. However, we can recreate the other features. Metasploit and ExploitDB are simply collected by seeing for which vulnerabilities the sources contain entries. The github feature is more complex as described in [19]. In this paper, they trained a machine learning model to predict whether GitHub repositories contained a working exploit code for a vulnerability. After manually labelling 800 repositories, the classifier was built and produce a prediction score for vulnerabilities. Some vulnerabilities might have no prediction score while others have many depending on the number of repositories it occurs in. This one to many mapping is changed to a binary feature by applying a threshold of 0.7.

EPSS is in the process of releasing the second version which has some major improvements to the prior model. In the first version of the model, EPSS only focused on recent vulnerabilities as it relied upon CVSSv3 which was only adopted from 2015. The newer version offers support for vulnerabilities with only CVSSv2 scores that date back to 1999. This improvement is realized by designing a machine learning model which maps CVSSv2 scores to CVSSv3.

In addition to the larger number of vulnerabilities supported, it also makes a significant improvement by including the lifetime of a vulnerability as a feature. This feature proved to be the ninth important variable in their model [19], indicating significant importance. Finally, EPSS is not just relevant for vulnerability prioritization, it also covers the subject of keyword extraction which we will elaborate on in the next section.

3.2. Keyword extraction

Keyword extraction is the technique of extracting desired information from (unstructured) data sources. Keyword Extraction is part of Text Mining research field which in turn falls within the domain of Natural Language Processing (NLP).

NLP research looks into how we can use textual analysis, pattern learning and pattern matching with a wide range of applications including finding synonyms [23] and extracting sentences. Not only is research varying in terms of the desired output, but also the way in which NLP is being used. Whereas some NLP research only considers the context directly next to a word, others take into account the entire document [35, 60].

While NLP can be used in different industries, some research has focused on the cyber security domain. Extraction of cyber security concepts like file names from text was performed in [31, 38]. In [29] multi-keyword extraction is used as described in [52] to retrieve useful information from the references present in CVE and NVD data. After scraping the referenced URLs, they extracted a list of tags that were used as features in the training of their model. Tags such as ”Code Execution”, ”Remote” and ”SQLi” proved to have a significant correlation with vulnerabilities being exploited in the wild. Similarly, in [32] they extract cyber security concepts and vulnerability descriptions from NVD and other text sources. These concepts and description can be added to the tages already found in EPSS [29].

Finally, research in keyword extraction has attempted retrieving STIX variables from unstructured threat advisories [49]. Ramnani et al. use a bootstrapping method [36] for extraction. The downside of this approach is that it requires a set of context patterns as input. Because only three cyber security advisories were used for their papers, this was not a problem. However, when considering that the NCSC uses more than 1.000 sources, defining these context patterns becomes a cumbersome task. Furthermore, only semantic and lexical scoring is used for advisories. As their goal is to best extract information these scores metrics make sense, however, as we desire filtering and prioritization of advisories as well other metrics and models are better suited.

3.3. Patch Management

Patch management is the process of identifying, acquiring, testing and installing patches for systems used by an organisation. Vendors release patches with the aim to resolve bugs, closing security holes or adding features to a product. Organisations need to monitor these publications and decide whether to apply them. Hence, patch management overlaps to some extent with vulnerability prioritization. While vulnerability prioritization focuses on selecting which vulnerabilities to remediate, patch management focuses on the application of patches to remediate the vulnerabilities. Research in the field provides insight into how we could automate cyber security advisories while still allowing organisations to efficiently apply patches.

Some research in patch management focuses on the timing of publishing patches [5, 13, 33] and the timing of applying patches [9]. The results presented in these papers, could be used to create features for our dataset. In combination with the time that an organization needs to notice that there is a relevant patch, this provide insight into how long it takes in total to remediate vulnerabilities. Our research reduces the noticing time by faster publishing third-party advisories. We can understand its value by comparing it with the timing of publishing and applying patches. Other research describes the cost of patching [6, 14], format of patches [50, 61] and the risks of not patching [15, 21, 54]. Analysts sometimes use these features when deciding which advisories to publish. Hence, these features could be a valuable addition to our model.

4 Methodology

The goal of this research is to develop a dataset that enables automating the collection and publication of advisories. To build this dataset we design a process that is based on the process currently performed by analysts. The process makes sure that dataset stays relevant as it can repeat the steps for the new vulnerabilities published each day. We propose this process in section 4.1.

The different stages of the process allow for procedurally answering the research question. The implementation of the stages is discussed in chapter 5 and chapter 6. Prior to covering those details, several decisions need to be made regarding how we will train and evaluate the model. In chapter 2, we covered the decision regarding which classifiers and training techniques to use. Besides these decisions, we also need to choose how to evaluate the performance of the machine learning model. We discuss the decision regarding the evaluation criteria in section 4.3.

4.1. Current process

Automating manual processes can be done in several ways. The easiest and most evident is taking the current steps of the process and automating each step. We apply this approach to the publication process of Dutch NCSC. Following an interview with one of the advisory creators from the Dutch NCSC, we find the following actions in the advisory publication process:

1. Collect vulnerabilities & advisories from sources
2. Gather extra information
3. Build advisory
4. Decide on publication
5. Review advisory
6. Publish advisory

Converting these steps to a flowchart results in the following diagram:

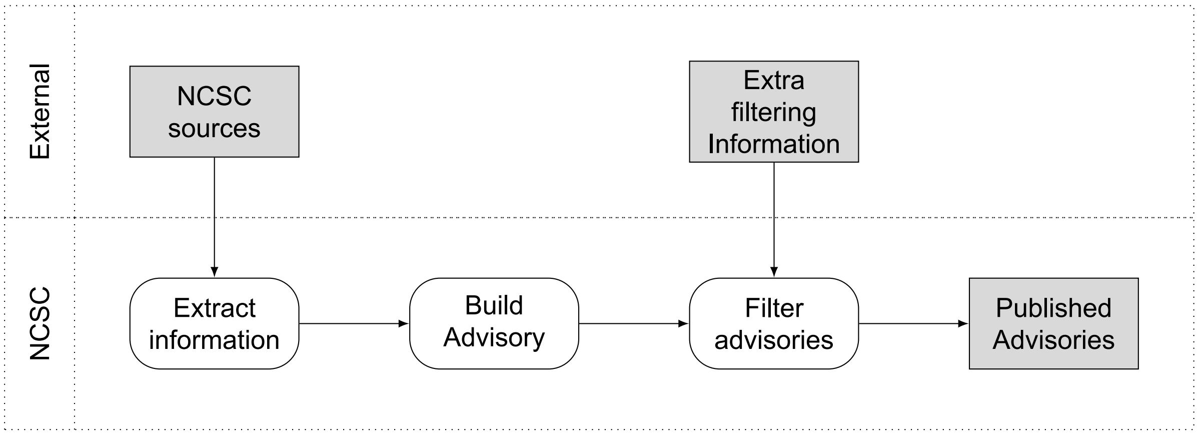


Figure 4.1: Current process

Each step in this model could then be automated to have an initial automated approach. However, this version of the process poses several problems. First of all, it is impossible to implement without the list of sources each NCSC uses. This list is often non-public data as it provides insight for hackers into which vulnerabilities might be more beneficial to attack. A vulnerability which does not appear in any of the sources used will also not receive an advisory. Hence, clients of the advisory publisher are less likely to have resolved the vulnerability making it more worthwhile for hackers.

Second, the current process is too simplistic and without a clear division between what work needs to be done by the NCSC and what is automated by this research. The current process also requires too many steps to be performed by the advisory publisher. In the end, we want to produce a process which requires them to do the bare minimum and is a generic solution for all advisory publishers. Hence, we make the necessary adjustments in the next section.

4.2. Proposed process

The first adjustment we make is to retrieve raw vulnerabilities rather than the NCSC sources. As explained in the prior section, we could not obtain the list of sources for any of the NCSCs so there it was not possible to implement the version depicted in fig. 4.1. An alternative approach that also provides a more general solution is by combining vulnerabilities ourselves. In this way, the set of created advisories is the same for an advisory publisher. We can retrieve the vulnerabilities from NVD, which we covered in section 2.4.2. We can combine the vulnerabilities retrieved from NVD to produce similar advisories to those published by the NCSCs. To mimic this behaviour correctly we need to study in-depth the different ways in which vulnerabilities are grouped together in advisories.

With this new approach, we might not be able to recreate all advisories as some combinations are infeasible to recreate. This could be due to certain combinations resulting in enormous data volumes which would decrease the performance of filtering later on. So with the new approach, we sacrifice a bit of performance but gain in terms of making a process that is generic and usable with just public data. We discuss several cases in which it was infeasible to recreate the combination of vulnerabilities in chapter 5. In the same chapter, we demonstrate the process of combining vulnerabilities and display how accurate the generated dataset is.

With the new process of automating advisories, we should also consider the fact that it is not a binary decision whether an advisory published is contained or not within our set. It could be the case that we combine an advisory such that it contains a percentage of the information but not all. In this case, it is not evident whether we mark the generated advisory as published or not. Hence, we develop a matching module with a scoring formula. The matching module computes the score between each generated advisory and published NCSC advisory. Then the advisory publisher can pick a threshold and if the generated advisory surpasses the threshold they are marked as published or not. The process of matching will be covered in-depth in chapter 5. The matching step provides the target feature for the machine learning model trained in the next step. In the filtering step, we use the target feature and all other features collected for generated advisories to train a machine learning model that predicts which generated advisories should be published.

All the adjustments made are visible in fig. 4.2. The new model can be split into two main steps; data collection and evaluation. The data collection phase is all the actions performed up to the matching module. At this point, the training set for machine learning is complete. Then in the evaluation phase, we train the machine learning model and evaluate the performance. This gives an indication of the extent to which the proposed process and created dataset are able to automate cyber security advisories. The two phases provide a clear segregation in this research and, hence, are covered within different chapters. In chapter 5, we cover the data collection and in chapter 6 we perform the evaluation.

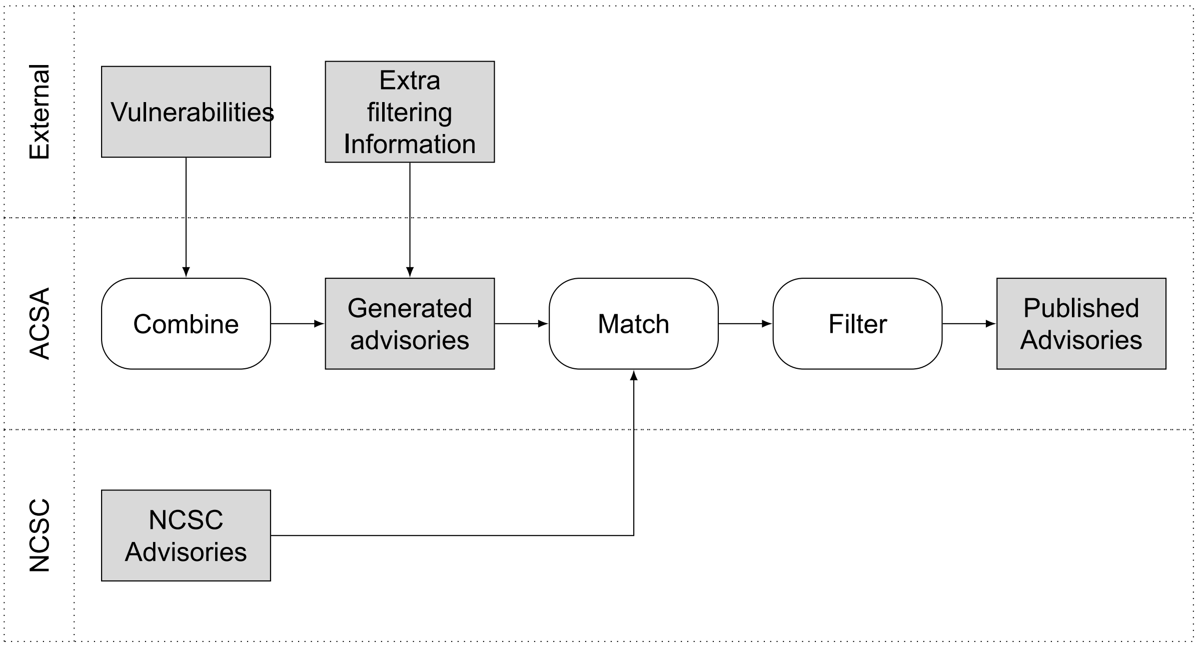


Figure 4.2: Initial process

4.2.1. Analyst feedback

The general version of the model looks promising for automating cyber security advisories. However, the model in its current state is non-dynamic. Consider the case in which the NCSC decides to change their policy regarding the publication of advisories. For example, clients switch between product vendors such that the NCSC should now publish advisories on the new vendor and potentially stop publication on the previous vendor. With the current model, it would take a long time before changes are applied to the model. This is due to the fact that the model still contains data which is now labelled incorrectly as the result of the policy change. Only when the new correct labelled data gains a significant share in the dataset, the model will adapt to the policy change. While the model is correcting itself, the performance of the model will decrease.

To minimize the performance reduction, an organisation has two options. First, relabelling all data such that it adheres to the new policy and then retrain the entire model. This option is worthwhile when a significant change is made to the policy, resulting in many labels that should change. Relabelling all data ensures that the model remains high performing. However, this method is labour-intensive with large datasets.

The second option is to use a feedback loop. A feedback loop, which was covered in detail in section 2.2.3, allows analysts to check a subset of advisories and relabel them manually. This solution is ideal when the change of labels is only small. In this case, a small feedback loop can be the more suitable solution as it allows the model to change continuously and, therefore, becomes more dynamic.

Nevertheless, there is also a downside to using a feedback loop. Introducing a feedback loop with humans performing the check manually could lower the performance if done incorrectly. Even worse, the controlling identities might be rogue, which could devastate the performance of the model. Keeping this in mind, we extend the prototype with the feedback loop:

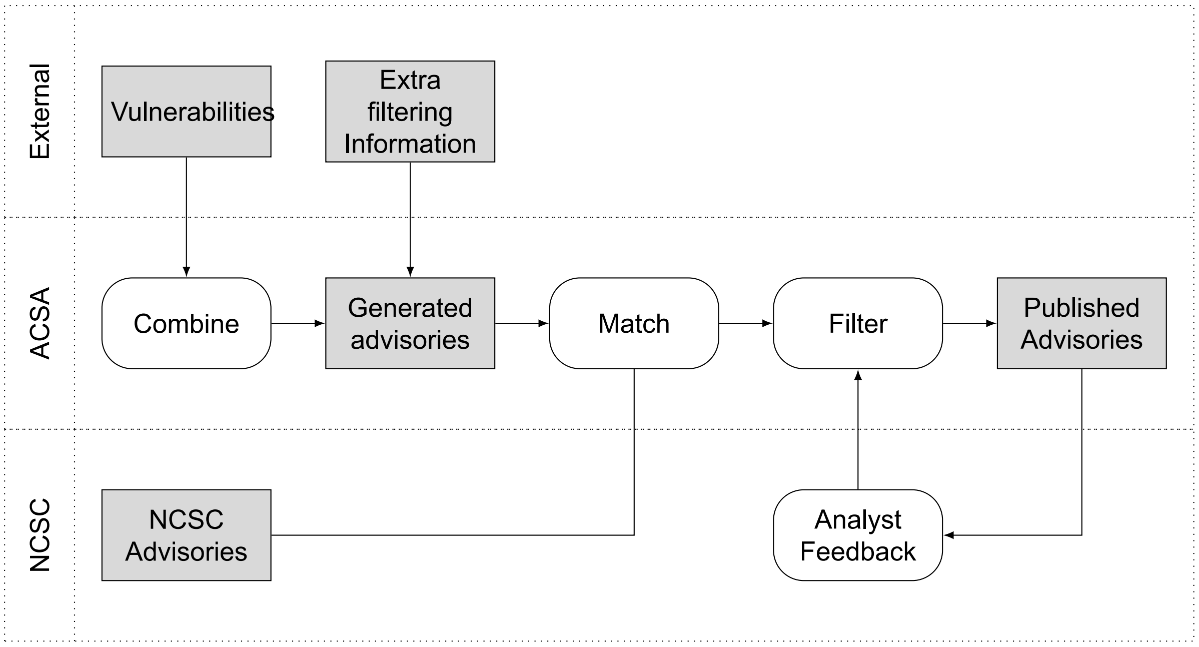


Figure 4.3: Improved process with feedback loop

4.3. Evaluation criteria

There are several techniques that allow for evaluating the model proposed. The selection of a technique is done with a focus on those who might benefit from using the model. The benefactors are organisations like NCSCs and other third-party publishers. These organisations can have different desires from the model. Where one publisher might want a low false-positive rate to reduce the workload, another chooses to maximize true positives to ensure few advisories go unnoticed.

Because the needs differ, we provide metrics that allow the publishers to see the performance of the model regardless of their difference in needs.

Confusion matrix

The confusion matrix provides the counts for the number of true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN). These labels show what the machine learning model predicts in comparison to what is actually expected. In this research, the four types of results can be described as follows:

- TP, a generated advisory is marked as published and is also published in the real world
- FP, a generated advisory is marked as published but is not published in the real world, this indicates a type 1 error.
- FN, a generated advisory is marked as non-published but it is published in the real world, this indicates a type 2 error.
- TN, a generated advisory is marked as non-published and is also not published in the real world.

This results in the confusion matrix looking as follows:

		Predicted	
		Published	Not published
Actual	Published	TP	FN
	Not published	FP	TN

Figure 4.4: Confusion Matrix

Accuracy

The accuracy represents the subset of all advisories correctly classified in the set of all advisories.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

Precision

The precision represents the subset of all advisories correctly classified as published in the set of all advisories classified as published.

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

Recall

The recall represents the subset of all advisories correctly classified as published in the set of all advisories which should be published.

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

ROC AUC

The receiver operand characteristic (ROC) curve plots the true positive rate against the false-positive rate. An example of a ROC Curve is shown in fig. 4.5. This graph allows picking the threshold for which the true positive to false positive ratio is best for someone's needs. The area under the curve (AUC) provides a metric which gives a more overall score of the model by computing the area under the ROC curve. The higher this score, the fewer false positives there are in comparison to true positives. Downside of ROC curves is that for classifiers which produce only labels without scores, the ROC curve contains just a single point. This is the case for the decision tree and random forest classifiers and, hence, we use f1-score metric to be able to compare the scores of these classifiers to the rest.

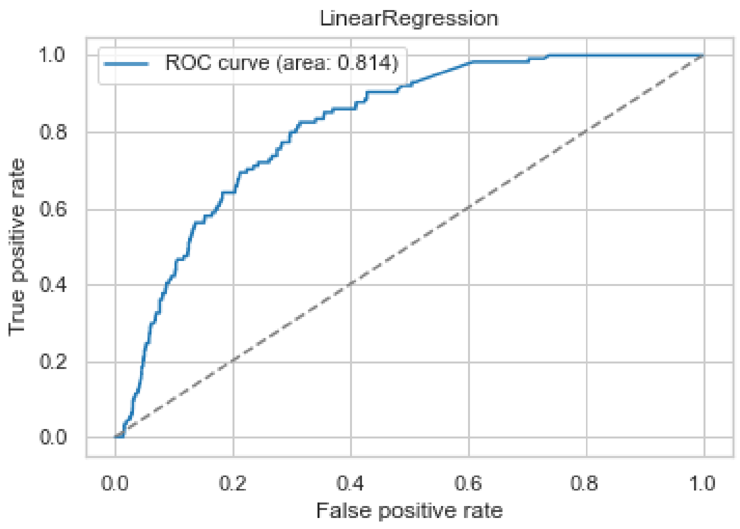


Figure 4.5: ROC Curve with AUC score

F1-score

The F1 score combines the precision and recall into a single metric by taking their harmonic mean. Due to this property, it is useful to compare the performance of classifiers. Where one might have a high precision, the other might have a higher recall making it difficult to establish which of the two is better. The F1 score makes this process easier.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{4.4}$$

Computational complexity

Besides metrics indicating performance with regards to labelling, we also look at the computational complexity. The different selected machine learning techniques will be compared to the others in terms of run time. Although our proposed model is trained offline, it might be retrained due to policy changes as discussed in section 4.1. Similarly, the model might be retrained every once in a while to include new advisories into the model. The computational complexity for the different classifiers is shown using both Big-O notation and average time to train the classifier. The Big-O notation represents the upper bound of the running time of an algorithm by measuring the worst-case scenario of an algorithm.

5 Data Collection

Now that the proposed process has been defined, the implementation of each of the steps can begin. The first actions are collecting the vulnerabilities and published advisories. In the background section, we already discussed all the data sources. Now, the information needs to be extracted. For efficiency, we develop a scraping module. First, we cover this scraper module that deals with the various data formats. Over the years, the industry has started to use standards more and more. Standards like CVRF and STIX make sure that data between different sources is represented in the same way, which allows for easy collection of data. For example, both Microsoft and Cisco present their vulnerabilities in CVRF format. Unfortunately, other sources still use custom JSON or HTML formats.

5.1. Data collection method

For each of the sources, we first need to retrieve the data. This is done using the python requests module[1] to fetch the website, which is multi-threaded for efficiency. Having received the data, we process it using the correct library depending on the data format. However, to know what should be included within both these classes we first need to cover how the matching between the classes will be done to find the target variable. This provides us with a list of variables that are mandatory for both classes. and then discuss how we use the module to collect the data. We split the data collection into advisories publishers and sources used for generating advisories.

5.2. Matching

Before we collect all the data, we need to determine what data is required for matching published and generated advisories. The matching uses a scoring function that indicates how similar generated advisories and published advisories are. For each generated advisory, we find the published advisory with the highest score and mark this as the best matching advisory. If the highest score meets the threshold specified by the advisory publisher, we mark the generated advisory as published and otherwise as non-published. This binary value will be used as our target feature in the machine learning model. The target feature is the actual outcome that we try to predict, which makes it an important component of this research.

[1]Version: 2.28.0

For the scoring function, we will compare the CVE IDs in a generated advisory versus those in a NCSC advisory. The CVE IDs are present in all advisories. Many other features that could be used for matching are embedded in the vulnerability linked to the CVE ID. These include date, products affected, vendor and tags. We have investigated using these features in the scoring function. However, this does not improve the matching mainly due two reasons. First, some features are more prone to inconsistencies for instance date. ... Second, some features like vendor are too generic and, therefore, do not improve the matching. Adding these features makes the scoring function unnecessary complex and, hence, we only use the CVE IDs.

Using, CVE IDs only we arrive at the scoring formula in eq. (5.1). This formula calculates the number of vulnerabilities that both the generated and published advisory have in common and divides this by the number of all unique vulnerabilities seen in both advisories. Any two advisories with the exact same vulnerabilities, receive a score of 100%. The score decreases when a vulnerability is missing in either advisory.

$$Score = \frac{|\text{Published Advisory CVEs} \cap \text{Generated Advisory CVEs}|}{|\text{Published Advisory CVEs} \cup \text{Generated Advisory CVEs}|} * 100\% \tag{5.1}$$

As mentioned prior, the applicant of the model can define the threshold for the score. In the ideal scenario, all published NCSC advisories match perfectly with an advisory in the generated set. This means that the list of CVE IDs in both advisories are identical. However, in reality this perfect match is hard to achieve. Analysts building the advisories might miss a vulnerability or use unstructured sources which combine vulnerabilities in a way that can not be perfectly reproduced with the data we have collected. Hence, it might be wise to use a lower threshold such that near perfect matches are also marked as published. In section 5.4.1 we further investigate how the threshold impacts the number of advisories marked as published. This provides insight for advisory publishers into which threshold is optimal for their desires.

Using the formula and threshold, we try to match published and generated advisories. However, external factors can influence the matching result and sometimes result in mislabelled data or missed matches. First, human errors in published advisories can cause some vulnerabilities to be missing or incorrectly added to an advisory. These vulnerabilities lower the matching score and may lead to an advisory incorrectly being marked as not published. Second, combining vulnerabilities in non-obvious ways will result in published advisories that are almost impossible to reproduce. Hence, these advisories are not present in the dataset, lowering the performance of our generating process. An example is vulnerability NCSC-2022-0121, which combines similar products of the same vendor. With the data we have collected it is not possible to find similar products and as a consequence we cannot generate a matching advisory.

5.3. Advisory publishers

In this paper, we evaluate our model using two different sources; the Dutch NCSC and the Canadian NCSC. In this section, we elaborate on how we collected the data for these sources. This demonstrates how another organisation could apply this model as well. Adding an organisation is made easy by performing the matching only the CVE IDs. As a result we only need to collect all the vulnerabilities in each advisory published. In the coming section, we sometimes also collect further information as it is used for the plots in section 5.6.

5.3.1. NCSC-NL

Retrieving the advisories from the Dutch NCSC is done via an open API. The API allows for retrieval of all advisories published in JSON, plain text and XML formats. As we are interested in the state advisories were in when they were published, we fetch the first version for each advisory. An example of a query is:

https://advisories.ncsc.nl/rest/advisory?ref=NCSC-2014-0002+%5B1.00%5D&format=htmlified

This URL retrieves the first version of advisory NCSC-2014-0002 in JSON format. Repeating this query for each NCSC ID, we retrieve all advisories. For each advisory retrieved, we select the 'cve_ids' fields which contains the list of CVE IDs in the advisory. Next to the CVE IDs, we collect the fields publication date, affected products and title of advisory. These fields are used in chapter 6, to find how the machine learning model behaves different throughout the years or highlight the difference between vendors or even products.

5.3.2. NCSC-CAN

For the Canadian NCSC it is more difficult to retrieve the information as it does not provide a disclosed API. Looking closer at the inner workings of the website, we discover calls to an URL that acts like an API:

https://cyber.gc.ca/webservice/en/v1/alerts

This URL returns a list of cyber security advisories published by the Canadian NCSC. Unfortunately, the information in the response is limited to the alert title, publication date and reference to an internal URL. So, to retrieve all necessary information we need to query the reference URL for each advisory. As these reference URLs point to web pages, the data is retrieved in HTML format. Observing several reference URLs, it becomes clear there are two default formats in which alerts are published; advanced and simple.

The advanced advisory provides a detailed analysis from the NCSC and contains all desired information. An example of such an advanced advisory is "Spring Remote Code Execution vulnerabilities" [43], which shows the structured format. We extract the sections: Audience, Purpose, Overview, Details and Suggested Action. These sections contain a lot of information; products affected, publication date, vendor, CVEs and tags. However, extracting these from unstructured plaintext requires a significant amount of work involving NLP. Furthermore, with our current matching formula we don't require these values. So, for now, we only extract the publication date and CVEs using a simple regex:

$$CVE - \backslash d\{4\} - \backslash d\{4,7\}$$

In the future, this could be replaced with a more advanced method, which will be elaborated in Chapter 8. These more detailed publications are referred to as alerts by the Canadian NCSC.

The simple advisories solely contain the publication date, products affected and reference URLs. These URLs refer to vendor advisories, which are even less structured. Similar to the advanced advisories, we try to extract the publication date and CVEs. As the publication date is already found, we continue with the CVE extraction from the reference URLs. Again using the CVE regex, we extract the CVEs.

Before applying the regex two extra measures need to be taken. First, not all reference URLs are of interest as some point to an index page containing all cyber security alerts like [58]. To build an exclusion list, we count the occurrences of all URLs. For all URLs occurring more than once, we manually verify whether it is an index page or not.

Second, some reference URLs point to monthly vulnerabilities reports like in the advisory [45]. The URL inside the advisory links to all release notes for Microsoft Edge over the past decade. Hence, we need to select a subsection of the webpage. Fortunately, the reference URL points to the correct location on the webpage using the URL fragment identified by the hashtag. This allows for extracting only the section of interest from the webpage. For URLs which do not contain a fragment identifier, we select the HTML body element to get rid of any redundant information. Next, we can apply the regex for CVEs over the extracted text and find the CVEs. For the 2000 advisories published, we find that only 1513 contain one or more CVEs as some advisories focus on threat actors or TTP. For these advisories, we have collected the minimal information required for matching advisories which are the CVE IDs but also the publication date.

5.4. Generating advisories

Before the matching of advisories can be done, we need to create a large dataset of advisories. The generation of advisories is done by combining vulnerabilities, which we first need to collect. The industry standard dataset for vulnerability information is NVD, which was already discussed in section 2.4.2. NVD contains all vulnerabilities and for each vulnerability it provides CPE, CWE and CVSS records. We scrape all this information for each vulnerability as these different types of information will be used for combining the vulnerabilities but also as features for the machine learning model.

Another approach to collect vulnerabilities would be to use a threat intelligence platform like IBM X-Force [26] or Rapid7 [51]. These platforms report on several different cyber topics including vulnerabilities and advisories. We scrape these platforms as well as they contain extra valuable information on vulnerabilities such as CVSS temporal metrics, which is not incorporated into NVD. Hence, we also scrape these sources. The threat intelligence platforms are used more for extra filtering information than for the advisory generation. However, they also allowed to confirm whether the vulnerabilities scraped from NVD where the exact same, which was the case.

5.4.1. Combining

Although a set of advisories was obtained, it can not yet be used for matching to the publisher advisories. Currently, the set of generated advisories is essentially all published vulnerabilities. However, advisory publishers combine several advisories and vulnerabilities to produce a single advisory. To mimic this behaviour, we first study the way both NCSCs combine vulnerabilities. After studying the behaviour, we implement it to produce the same advisories. Next, by applying the matching module we determine whether the combining was sufficient and study cases that were not properly matched.

Browsing through the publications, a clear pattern of how vulnerabilities are combined becomes visible. For example, looking at the publications on the 8th of march, we see the publications NCSC-2022-0156, NCSC-2022-0157, NCSC-2022-0158, NCSC-2022-0159, NCSC-2022-0160. Each of these publications covers a product group from Microsoft and is published on the same date. The product groups respectively are Azure, Office, Exchange, Defender and Windows. Having identified a way to combine vulnerabilities, we check manually if it is possible to recreate the advisories. Picking the advisory NCSC-2022-0158 which is for Microsoft Exchange Server, we find it contains CVE-2022-23277 and CVE-2022-24463. Recreating the advisory is done best using the API from NVD which provides many ways of selecting vulnerabilities [46]. We query the API using the following parameters:

Parameter	Function	Value
cpeMatchString	Filtering affected products	cpe:2.3:*:microsoft:exchange_server:*:*:*:*:*:*:*
pubStartDate	Filtering publication date	2022-03-07T00:00:00:000 UTC-01:00
pubEndDate	Filtering publication date	2022-03-09T00:00:00:000 UTC-01:00

Table 5.1: NVD API parameters for NCSC-2022-0158

The cpeMatchString uses the CPE name matching convention, which was covered in section 2.4.2, to filter affected products. The value used retrieves all vulnerabilities for all Microsoft Exchange Server products and versions. The other values are used to narrow down the publication date. A window is used as there may be a difference between the publication date from the NCSC and the date in NVD, which is also observed for CVE-2022-23277. This difference can be due to an organisation receiving non-public information or using other sources which publish earlier than the NVD. Similarly, there can also be a delay between when the NCSC publishes due to having to review the vulnerabilities and deciding on publication. Hence, the window of two days before and two days after the NCSC publication. Using this window, the query finds just both the required CVES: CVE-2022-23277 and CVE-2022-24463. Meaning the advisory was successfully recreated. We can apply this way of combining all advisories by creating a bucket for each product found. To each bucket, we add advisories which match the beginning part of the CPE string until the product. So any two advisories which contain the CPE string cpe:2.3:X:X:X:*:*:*:*:*:*:* are combined in the same bucket. Finally, we combine all CVEs of the advisories within a bucket to create a new advisory. This methodology yields the following results depicted in the table below and causes the dataset to contain 60,132 advisories in total. The generated percentage is calculated by the number of advisories divided by the total number published for each organisation, which is 8,574 advisories for the Dutch NCSC and 1,513 for the Canadian NCSC.

Threshold	Advisories	Generated %
>=100%	4,757	55.3%
>=90%	4,944	57.5%
>=70%	5,479	63.7%
>=50%	7,053	82.0%

Figure 5.1: Alerts NCSC NL

Threshold	Advisories	Generated %
>=100%	587	38.8%
>=90%	604	39.9%
>=70%	658	43.5%
>=50%	1,024	67.6%

Figure 5.2: Advisories NCSC CAN

Seeing the high percentage of advisories scoring less than 50% of CVEs in its advisory correct, indicates that there are more ways in which vulnerabilities are combined. Studying these cases, we identify advisory NCSC-2022-0099 and NCSC-2022-0124. These advisories do not just contain vulnerabilities for a single product from a vendor but for all products from a vendor. We manually recreate the advisory from NCSC-2022-0124 using the NVD API with the following parameters:

Parameter	Function	Value
cpeMatchString	Filtering affected products	cpe:2.3:*:cisco:*:*:*:*:*:*:*:*
pubStartDate	Filtering publication date	2022-02-20T00:00:00:000 UTC-01:00
pubEndDate	Filtering publication date	2022-02-30T00:00:00:000 UTC-01:00

Table 5.2: NVD API parameters for NCSC-2022-0158

The query is similar to one previously used, except the CPE string now also contains * selector for the product. Applying this way of combining all advisories yields the results displayed in the table below. We see a relatively small increases in the size of total dataset grows to 64,085, but significant increases in the number of perfect matches.

Threshold	Advisories	Generated %
>=100%	5,905	68.7%
>=90%	6,130	71.3%
>=70%	6,737	78.3%
>=50%	8,430	98.0%

Figure 5.3: Alerts NCSC NL

Threshold	Advisories	Generated %
>=100%	714	47.2%
>=90%	752	49.7%
>=70%	851	56.2%
>=50%	1,298	85.7%

Figure 5.4: Advisories NCSC CAN

Finally, we identified that with the NVD data we scraped there were many reference URLs present. By comparing the reference URLs to those present in the advisories published for both NCSC, we found there was a significant overlap between the URLs. This indicated that it might be worthwhile to scrape all the reference URLs and extract the CVE IDs from each page. The combinations made in these URLs are sometimes very specific as is the case in the NCSC-2022-0545 advisory. This advisory combines several vulnerabilities from across several years based on whether they are remediated in a certain patch. We cannot recreate the same combination and, hence, such an advisory receives a low score. To improve these advisories, we scrape each reference URL and apply the same regex as used for the retrieving of CVE IDs previously:

$$CVE - \backslash d\{4\} - \backslash d\{4,7\}$$

After scraping more than a million URLs, we found that the improvement was significant. Unfortunately, the downside of this is that it increases the number of advisories in the dataset significantly. This could result in a lower filtering performance, which in turn could result in a lower automation score despite the improved matching scores. This effect will be studied in chapter 6. In the end, using all the different combination techniques we have created 88,428 advisories. In all these advisories, we have perfect matches for 82% of the Dutch advisories and 61% of the Canadian advisories. This would allow to directly automate the majority of advisories for both organisations if the filtering would work perfect. The score can even be higher, if we lower the threshold of the matching score.

The three different methods of combining do create some similar advisories. We see that across the three different combining methods the overlap between the advisories is no more than 10% between each combining method. Hence, each method is required to obtain the results in the table below and cannot be optimised any further. The exact duplicates are filtered out and the ones which are almost the same are left in the dataset. This results in the fact that some published advisories might have one or more generated advisories marked as published. Filtering out the near-duplicates is difficult and as much of the data stays similar the impact on the performance is low. We do note that in this way we are oversampling some of the records but as it happens for all records the impact is not significant. For all the machine learning model training, we select the threshold of generating 0.9. This value provides a good balance between allowing for errors and still requiring a good match.

Threshold	Advisories	Generated %
>=100%	7,013	81.5%
>=90%	7,064	82.1%
>=70%	7,483	87.0%
>=50%	8,560	99.5%

Figure 5.5: Alerts NCSC NL

Threshold	Advisories	Generated %
>=100%	924	61.0%
>=90%	1,003	66.2%
>=70%	1,120	74.0%
>=50%	1,398	92.3%

Figure 5.6: Advisories NCSC CAN

5.5. Extra filtering information

To filter a dataset a machine learning model requires a set of features. At the moment, we only have the target feature if we select our threshold for matching. In the coming sections, we add many different features next to the target feature and elaborate on the process required to add the features. The features are grouping into CVSS, Tags, Exploit status and custom features.

5.5.1. CVSS

Using the list of CVE IDs in each advisories, we can scrape extra data from the NVD. The NVD provides CVSS scores and CWE data, both useful for machine learning features. CVSS consists of two main vectors, the base vector and temporal vector. For each of the vectors a formula is used to produce the respective base and temporal scores. For these scores, we compute the average, maximum and minimum across all CVEs in an advisory. For the individual features in the vectors, we binarize all possible values. Next to binarizing, we can also compute the average, maximum and minimum for all individual features in the vector. The base vector contains eight features and three for the temporal vector. However, the vector values are non-numeric values. So, prior to applying mathematical functions to the values, we translate them to numeric values. This is done using the constants used in the CVSS formula. The values for each of the attributes are found in the table A.1, which originates from [18]. Instead of using the constant values, we can apply a custom mapping like label encoding. This mapping, however, is not well balanced unlike the constant values from the CVSS formula. These values have been balanced, to ensure an even distribution of CVSS scores. After, translating the values to numeric, we compute the average, maximum and minimum for each vector metric. Unfortunately, the temporal scores are not included in NVD. Hence, we use IBM X-force which includes the temporal vector and score. In the end, we have collected 39 features shown in table A.1.

Metric	Number of features	source
Base score	3	NVD
Base vector	24	NVD
Temporal score	3	IBM X-Force
Temporal vector	9	IBM X-Force

Table 5.3: CVSS metric features

5.5.2. Tags

Next to CVSS data, the NVD dataset provides CWE data. CWE data facilitates the type of weakness the vulnerability targets, which we refer to as tags. CWE also groups the weaknesses into categories and views as was covered in chapter 2. The categories and views make the maximum number of features 1324. The distribution of these features are shown in table 5.5. Having this many features makes the training of machine learning model unnecessary slow and, hence, we filter features.

The filtering process starts by removing the views. Views map weaknesses into different categories depending on the usage scenario. As we are conducting a research, we select the Research Concepts view to group categories and weaknesses and discard the other views. The research view organizes items by their behaviour using multiple levels of abstraction, which suits this research.

When the occurrence of a feature is too scarce, the added performance is low while it increases the run-time. Therefore, we filter out those features occurring sporadically in advisories. With regards to weaknesses, this is done by selecting only the 25 most occurring weaknesses as listed by MITRE each year [36]. Similarly, we filter out categories by only selecting the top 100 categories.

The tags from the NVD references are extracted using the method described in [29]. EPSS extracts multi-word expressions from the reference URLs present in each CVE. The extraction is done using Rapid Automatic Keyword Extraction [52] and results in a normalized list of 191 tags. The creators of EPSS provided the direct mapping of CVE IDs to tags, allowing for easy application. Using the mapping on the CVEs in each advisory, we obtain a list of tags for each advisory. Selecting all the unique values in the list, we find 175 features.

Metric	Features available	Features selected
Weaknesses	926	25
Categories	351	100
Views	47	0
EPSS	191	175
Total	1,515	300

Table 5.4: CWE features

5.5.3. Exploit status

Organisations will publish an advisory whenever a vulnerability is being exploited to increase awareness and prevent hacks from occurring. Vice versa, if there is a lower risk, due to no exploitation or PoC, organisations might opt to not publish an advisory. Expecting a correlation with the exploit status, we try to expand our model with some of the features used by EPSS. The features we duplicate are Exp:GitHub, Exp:Metasploit and Exp:ExploitDB. These are some of the highest ranked features for EPSS as shown in [19] and, therefore, promising features for this research.

A feature for GitHub was received from EPSS as implemented in [19]. In this paper, they trained a machine learning model to predict whether GitHub repositories contained a working exploit code for a vulnerability. After manually labelling 800 repositories, the classifier was built and produce a prediction score for vulnerabilities. Some vulnerabilities might have no prediction score while others have many depending on the number of repositories it occurs in. This one to many mapping is changed to a binary feature by applying a threshold of 0.7, which was the value also used by EPSS.

Both Metasploit and ExploitDB are more simple features in contrast to GitHub. Both datasets contain entries for which vulnerabilities contain working exploits, which we can directly map to two binary features. Both datasets are small, however, they do provide a good indication of exploitation activity.

Metric	Features available	Features selected
GitHub	1	1
Metasploit	1	1
ExploitDB	1	1
Total	3	3

Table 5.5: Exploit status features

5.5.4. Custom

The last features we add to the model, we have designed ourselves inspired by EPSS. A feature used by EPSS is the number of references present in NVD. Like CVSS, we cannot use this features directly as our advisories often contains multiple vulnerabilities. Hence, we repeat the process of computing the average, min and maximum.

Another feature we add is the number of vulnerabilities within a generated advisory. This feature provides context to many other features as we reduce them from a set to a single value. Besides the number of CVE IDs in an generated advisory, we use two other feature sets that are generated by the combining of vulnerabilities. The first feature is the vendor for which all the vulnerabilities are grouped. If an advisory is not created using a grouping on vendor the value for this feature is none. Similarly, we created a feature for when vulnerabilities were combined for a specific product group. For vendors we selected the top 150 vendors whereas for the product group we select the top 50 most occurring products. Below both thresholds, the number of records in the dataset containing the feature would be so small that the feature doesn't impact the filtering. Finally, we obtain the set of custom features described in table 5.6.

Metric	**Features**
Number of References	1
Number of CVEs	1
Vendor	150
Product Group	50
Total	204

Table 5.6: Custom features

5.6. Data analysis

To get a feeling for all the data created, we visualize the data with graphs and tables. fig. 5.9 shows the distribution of published advisories by the NCSC-NL. From the figure it becomes clear that the number of publications remains fairly constant since the first publication. This is quite remarkable considering that the number of known vulnerabilities has increased rapidly over the years. Advisories are published on vulnerabilities, hence we would expect a correlation between vulnerabilities and advisories published. This could indicate that the NCSC has changed its decision process for publishing advisories or strives to keep the number of publications below a certain number. In either case, this causes the model to have a lower performance.

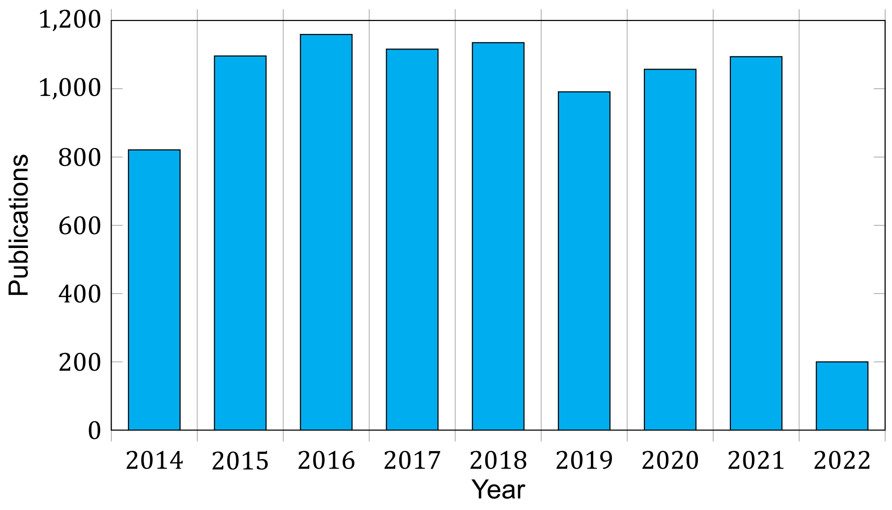


Figure 5.7: Advisories NCSC NL

The distribution of NCSC-CAN advisories correlates better to the number of vulnerabilities. The number of advanced advisories decreased in 2021, which is remarkable. However, these only account for less than 5% on a yearly basis, making it is less significant.

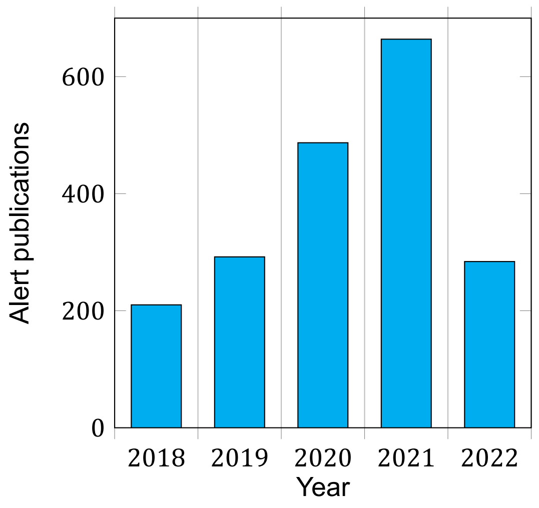


Figure 5.8: Simple advisories NCSC CAN

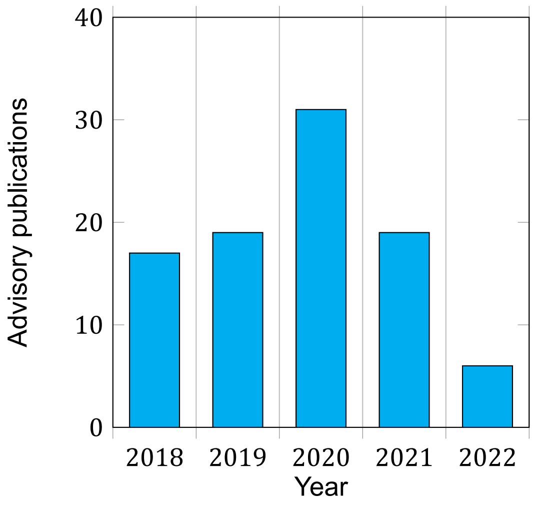


Figure 5.9: Advanced advisories NCSC CAN

Finally, we can analyse the performance of features. First we can list the top vendors for which the NCSCs publish advisories. This gives an impression on how valuable the vendor feature might be. If for instance a large percentage of advisories is published on a single vendor, this vendor will be an important feature for the model.

Index	Vendor	Advisories	Cumulative %
1	Microsoft	701	8.2%
2	Cisco	606	15.2%
3	IBM	563	21.7%
4	Oracle	325	25.5%
5	Google	252	28.5%
6	Adobe	235	31.2%
7	Apple	228	33.8%
8	F5	191	36.1%
9	Red	174	38.1%
10	Apache	151	39.8%
20	Fedora	85	51.7%
30	OpenSUSE	58	60.2%

Table 5.7: Data NCSC-NL

Index	Vendor	Advisories	Cumulative %
1	Cisco	174	11.5%
2	Microsoft	136	20.5%
3	Google	133	29.3%
4	IBM	108	36.4%
5	Adobe	105	43.3%
6	Mozilla	99	49.9%
7	Apple	81	55.2%
8	Ubuntu	68	59.7%
9	Siemens	65	64.0%
10	HP	58	67.8%
20	Citrix	26	85.1%
30	Palo Alto	13	90.1%

Table 5.8: Data NCSC-CAN

Next to the vendor feature, we can show the correlation of all features to our target feature. As we have a large number of features, we only display the features with a correlation larger than 0.1 in the table below.

Feature	Correlation score
vendor:Microsoft	0.193961
cvss:RL:Unavailable	0.160188
cvss:RL:Official Fix	0.159733
cvss:RC:Confirmed	0.147360
vendor:Adobe	0.104914
tag:memory corruption	0.101412

Table 5.9: Correlation features NCSC-NL

Feature	Correlation score
vendor:Mozilla	0.193174
vendor:Google	0.161361
vendor:VMware	0.139535
product:Firefox	0.136014
cvss:I:High	0.110949
vendor:Rockwell	0.108546

Table 5.10: Correlation features NCSC-CAN

6 Evaluation

In this section, we use the data collected to build and evaluate the classifiers discussed in section 2.3. We analyse the performance by plotting the different ROC curves for the models along with the F1-scores. Using the metrics, we compare the different classifiers. For the XGBoost classifier, we find the threshold that maximises the F1-score. Then we retrieve the confusion matrix that belongs to that threshold. The confusion matrix sketches a possible scenario of the number of true and false positives. This gives an impression of the extent to which the dataset can automate cyber advisories and how much work analysts still need to do to filter out the undesired advisories.

Further analysis of the model is done with SHAP. SHAP analysis allows to see the features which have the most significant impact on the outcome of the model. It also shows how the value of a feature relates to the predicted outcome of a model. This allows us to identify cases for which the model predicts a published or not published label. These cases are then analysed to find for which cases the final XGBoost model results in a true positive, false positive or false negative.

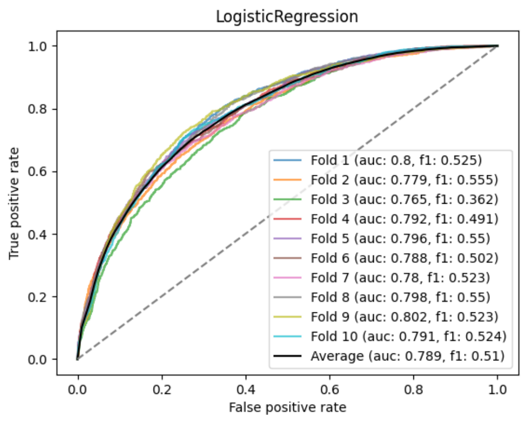
The aforementioned steps are done using all the features we have collected in chapter 5. However, some of the features change over time, which impacts the performance of the model. In section 6.4, we investigate this in detail. First, we display the various ROC curves along with AUC and F1-scores for the four classifiers chosen:

1. Logistic regression
2. Decision tree
3. Random forest
4. XGBoost

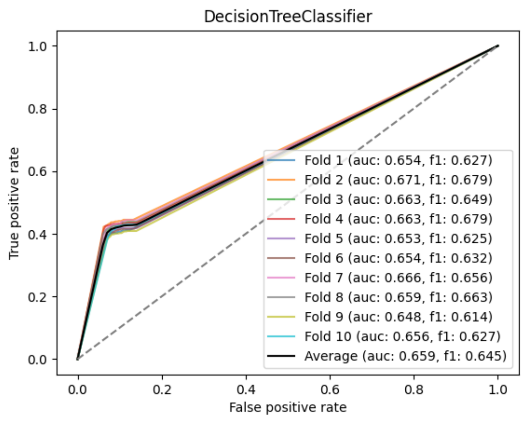
These classifiers are trained using all the features listed in section 5.5. For each machine learning model we build, we apply a stratified 10-fold. Ten is the default parameter and the value commonly used for stratified k-fold. This value has been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance [30]. In each plot, we show the different folds and the average ROC curve across all folds. Then finally we plot all the average ROC curves for all the classifiers in a single plot. This allows for an easy comparison of the performance. We first show the results for the Dutch NCSC and then for the Canadian NCSC.

6.1. NCSC-NL

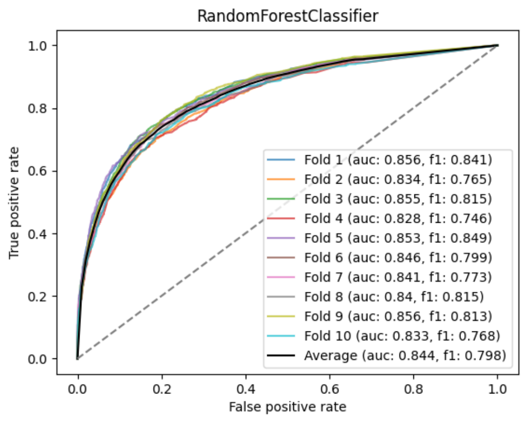
The various graphs in fig. 6.1 give a clear indication of the performance of the models. Across the 10 folds, there is only a small deviation between the ROC curves. Similarly, the ROC AUC has a small deviation as it is the area under the ROC curve. The scores differ only by a maximum of 2%, which indicates that the model performs consistently between different folds. Whether the model is also consistent in its choices of labelling should be analysed by comparing the different models. This will be done in the SHAP analysis section.



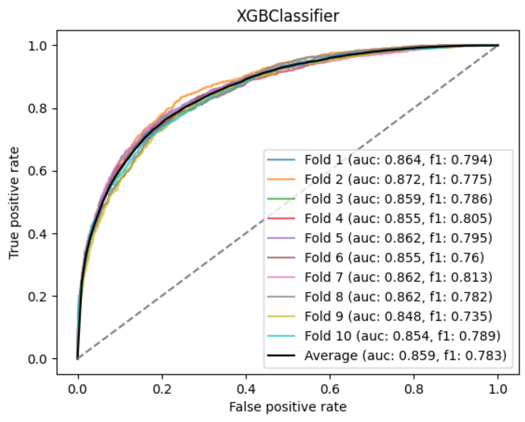
(a) Logistic regression



(b) Decision tree



(c) Random forest classifier



(d) XGBoost classifier

Figure 6.1: Classifier performance plots

Not only the deviation of ROC AUC scores is adequate, but the overall score is also promising. The higher the ROC AUC score, the better the model performs at classifying. The diagonal line displayed in the graphs shows the ROC curve of performing the task at random, which corresponds to an AUC of 0.5. This demonstrates that the model performs a lot better. However, it is not perfect because the score is lower than 1.0. This means that the model is not able to find all true positive cases while not creating any false positives. fig. 6.1d shows that we are able to have a true positive rate of 0.5 with a false positive rate of less than 0.1. Depending on the organisation's requirements, this could be acceptable for fully automating a part of the advisories. Recall that we generated 7013 advisories which perfectly matched with advisories from the Dutch NCSC. A true positive rate of 0.5 results in around 3500 advisories being published automatically. However, this would also publish false positives as the FPR is 0.1.

6.1.1. Classifier comparison

In the previous paragraph, we analysed the overall performance of classifiers. In this section, we will compare the different classifiers using fig. 6.2. This graph displays the average ROC curve for all the different classifiers. From the graph, the best-performing classifier can be seen. XGBoost has the highest ROC curve and also the F1-scores are almost the highest. Hence, we will use the XGBoost classifier for further analysis of the performance of the model. Nevertheless, logistic regression and random forest also have adequate performance, almost as high as XGBoost. Both have a ROC AUC of 3% less than XGBoost and the F1-score for the random forest classifier is even higher than for XG-Boost. Hence, organisations can also use the other classifiers if they better suit their needs.

The decision tree classifier is performing significantly worse than the other classifiers. The decisions made by an organisation to publish advisories are too complex to be mapped inside a single decision tree. The decision tree is not refined enough, which results in many false positives. Similar behaviour is observed when looking at the boosting process of XGBoost and random forest. When using one or two trees, these classifiers also have a low performance. In the coming sections, we will only use XGBoost to further analyse the performance as it has the highest performance and fast computational time.

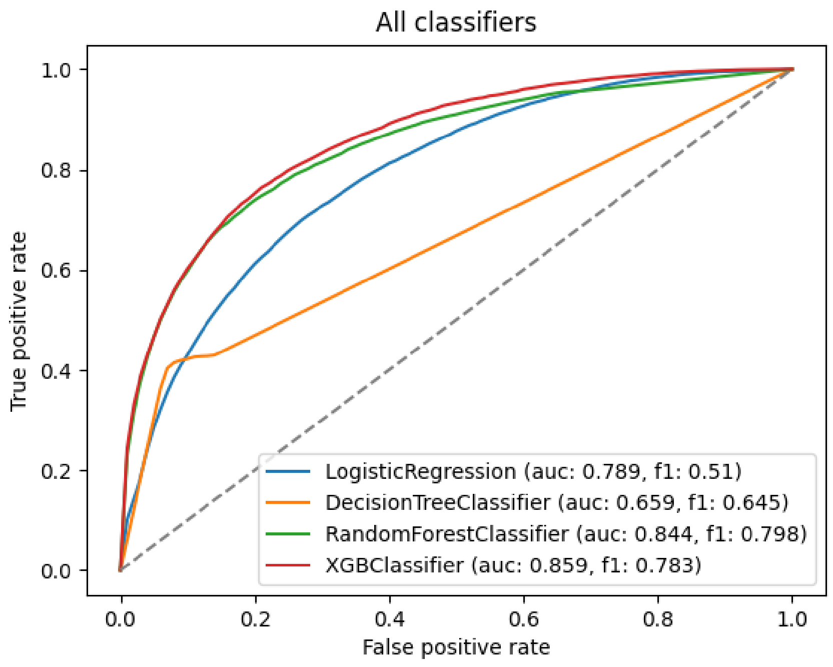


Figure 6.2: Average classifier ROC curve

6.1.2. Detailed analysis

To study the number of false positives we find the confusion matrix that corresponds to the maximal F1-score. In fig. 6.3 we plot the F1-score against the threshold. This threshold maps the predicted probabilistic value to the final label. A threshold of one results in no advisories marked as published. The lower the threshold goes the more advisories will be published, which also increases the false positive count. From the graph, we find the optimal f1-score which corresponds to a threshold of 0.17. Applying this threshold, we can retrieve the confusion matrices for all the non-overlapping folds. We studied the models between the different models and found that each model outputs almost the decision which are important. This behaviour is further studied in the SHAP analysis later on. Finally, we add the scores of each confusion matrix to retrieve the overall scores shown in fig. 6.4.

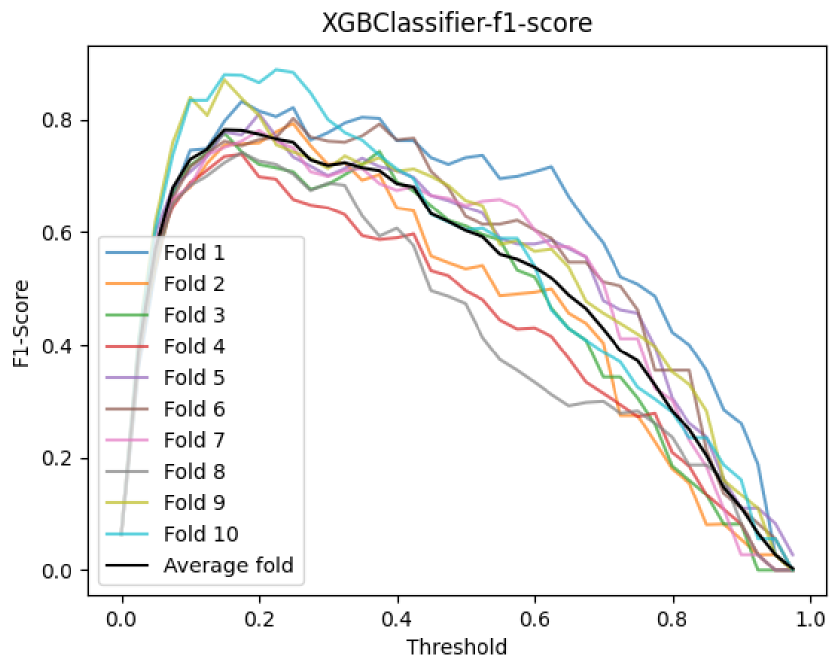


Figure 6.3: XGBoost F1-scores

		Predicted	
		Published	Not published
Actual	Published	4,593	2,420
	Not published	386	88,428

Figure 6.4: Confusion Matrix

From the table it becomes clear that we are able to automate more than 50% of the Dutch advisories. The number of false positives is quite low and can be filtered by analysts. This still reduces the workload compered to what is currently done by NCSC. From checking tens of thousands of sources to just checking a few hundred generated advisories. Next, we study whether we can further improve the scores by applying the model only on a selection of vendors. The organisations can also opt to only use the trained model on a selection of vendors or even already filter the vendors before training the model. We train a classifier on all vendors with a test size of 25%. In the test case, we find that the model performance deviates significantly between vendors:

Vendor	**TP**	**TP + FN**	**Recall**
Microsoft	259	280	0.925
Cisco	50	172	0.291
Google	48	56	0.85

Although this test case only uses a limited set of data, it illustrates that a selection of vendors might yield an increased performance. For example, we are able to find nearly all advisories that should be published for Google and Microsoft while for Cisco we miss many of the advisories. Hence, using the model on a selection of vendors could result in an even higher F1-score and ROC AUC. To verify this behaviour, we study the top 10 vendors in the dataset as shown in table 6.1. We summarize the true positives found in each fold and compare the total amount to the number of generated advisories that should be published.

In table 6.1, the performance of generating advisories and machine learning model is shown for the top 10 vendors. Combined they give us an indication of the extent to which we can automate cyber security advisories for specific vendors. For instance, there are 701 advisories published on Microsoft within the Dutch NCSC. From these 701 advisories, only 685 were created by our merging process. Then using a machine learning model and the optimal threshold selected using the F1-curve we find 665 out of the 701 published advisories. This in total results in the full model being able to directly automate more than 90% of the Microsoft advisories. Concerning the other vendors, we see vendors like Red Hat scoring much lower. The performance difference between the vendors is easily explained by looking at the difference between the advisories present for the different vendors. The advisories from Microsoft are very consistent with all advisories for Microsoft Exchange being published whereas other products are not published consistently. Leaving out vendors like Red Hat increases the performance of the model. We find that training the model only using the data of the top 10 vendors increases the AUC to 0.89 and F1-score to 0.84. Meaning the ratio of true positive with respect to false positive and false negatives improves. For a scenario analysis in this model, we obtain that we can automate 62% of all advisories with the same number of false positives.

Index	**Vendor**	**Published**	**Cumulative %**	**Generated**	**TP**	**Recall**	**TP/Published**
1	Microsoft	701	8.2%	685	665	0.97	0.95
2	Cisco	606	15.2%	513	115	0.22	0.19
3	IBM	563	21.7%	300	284	0.95	0.50
4	Oracle	325	25.5%	195	102	0.52	0.31
5	Google	252	28.5%	220	185	0.84	0.73
6	Adobe	235	31.2%	221	192	0.87	0.82
7	Apple	228	33.8%	202	183	0.91	0.80
8	F5	191	36.1%	182	173	0.95	0.91
9	Red Hat	174	38.1%	37	2	0.05	0.01
10	Apache	151	39.8%	139	116	0.83	0.77

Table 6.1: Vendor performance NCSC-NL

6.2. NCSC-CAN

The results for the Canadian NCSC are similar to the Dutch NCSC except for the Canadian average AUC which is 6% higher. This is a significant difference and means that we could have the same true positive rate while having less false positives. The increase of the AUC score could be due to decisions from the Canadian NCSC being more consistent for the features gathered. In the previous chapter, we observed a difference in behaviour between the two NCSCs, which could explain the difference in performance. The model is consistent across the different folds and the relative performance difference between the classifiers is also the same as can be seen fig. 6.5. The performance across the different folds for each classifier can be seen in appendix C.

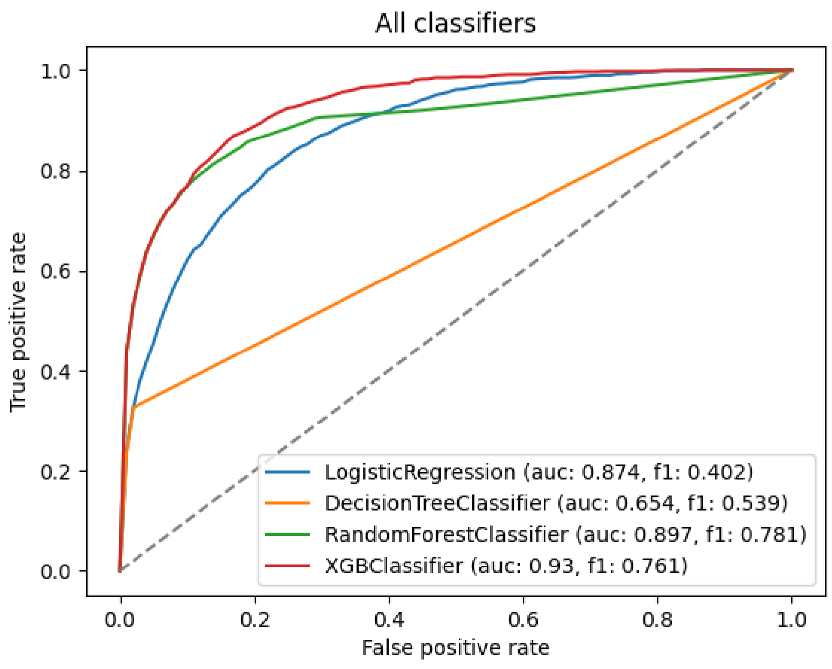


Figure 6.5: Average classifier ROC curve

6.2.1. Detailed analysis

Next, we analyse the confusion matrix that corresponds to the maximal F1-score for the XGBoost classifier. In fig. 6.6 we plot the average F1-score across all folds against the threshold. The threshold is the criterium which maps the predicted probabilistic value to the final label. The lower the threshold goes the more advisories will be published, which also increases the false positive count. From the graph, we find the optimal f1-score which corresponds to a threshold of 0.275.

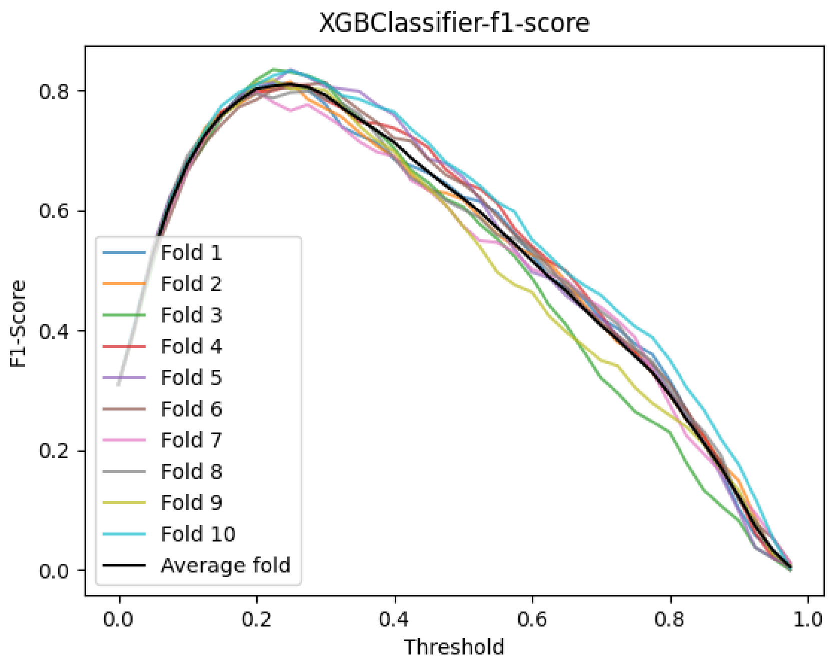


Figure 6.6: XGBoost F1-scores

Then we apply the threshold of 0.17 to find the confusion matrices for each fold. All the matrices are added up to retrieve the final confusion matrix, which gives an actual impression of the number of false positives. In fig. 6.7, we show this confusion matrix.

		Predicted	
		Published	Not published
Actual	Published	731	272
	Not published	140	88,285

Figure 6.7: Confusion Matrix

From the confusion matrix, we find that we are able to automate 731 of 1513 advisories from the Canadian NCSC while having only 140 false positives. It is important to note that although the confusion matrix shows we only missed 272 advisories. The actual count is higher as we selected a scoring threshold of 0.9. This resulted in only 1003 being matched in the dataset of 1513 NCSC advisories actually published.

Furthermore, we analyse whether the performance is higher when we look at individual vendors. In table 6.2, we see that there is a significant performance difference between vendors. We observed a similar pattern for the Dutch NCSC. For the Canadian NCSC, however, we see that the difference between large vendors is smaller. The performance of the model drops for smaller vendors. This is expected behaviour as the machine learning model trains on a dataset that is not balanced across vendors. The dataset only contains a few samples for the smaller vendors and, thus, it has a hard time creating good decisions for these samples. We could reduce this effect by oversampling the smaller vendors or filter out the smaller vendors. We perform the latter by training a model only on the samples of the top 10 vendors. As with the Dutch NCSC, we find a much higher

Index	Vendor	Published	Cumulative %	Generated	TP	Recall	TP/Published
1	Cisco	174	11.5%	159	126	0.79	0.72
2	Microsoft	136	20.5%	127	121	0.95	0.89
3	Google	133	29.3%	128	108	0.84	0.81
4	IBM	108	36.4%	63	58	0.92	0.54
5	Adobe	105	43.3%	86	65	0.76	0.62
6	Mozilla	99	49.9%	81	54	0.67	0.55
7	Apple	81	55.2%	78	72	0.92	0.89
8	Ubuntu	68	59.7%	43	22	0.51	0.32
9	Siemens	65	64.0%	48	23	0.48	0.35
10	HP	58	67.8%	41	22	0.54	0.38

Table 6.2: Vendor performance NCSC-CAN

6.3. Case studies

In this section, we analyse TP, FP and FN cases for both NCSCs. This analysis is quite complex as XGBoost adds the scores of all trees created during the boosting tree. A possible way to analyse cases would be to find all the individual scores and inspect the decisions that led to that score.

To do this, in each tree we could follow the decision path for the advisory to arrive at the leaf node for each advisory. The leaf node contains the raw score which is added for each tree. A positive score increases the chance of it being published while a negative score decreases it. We add all the scores for the trees sequentially to find the final score. Then we can convert the score from logistic to probabilistic scores using eq. (6.1). Using a chosen threshold, the values above the threshold are mapped to published and below to not published. This process gives insight into the exact decisions that led to an advisory being published or not. To illustrate, how this behaviour works we have included some of the decision trees in appendix B.

$$Probabilistic\_score = \frac{1}{1 + \exp(-1 * leaf\_value)} \tag{6.1}$$

However, this process is cumbersome. A simpler solution would be to use SHAP to analyse cases. SHAP shows which features have the biggest impact and how they impact the outcome. In other words, SHAP analyses the decisions that lead up to the score for each tree and weighs these decisions across all the trees to provide a summary. In this way, we can identify advisories which are likely to be published using only a few properties. This allows for an easier identification and analysis of TP, FP and FN cases.

Before analysing the cases with SHAP, we have to determine the number of boosting rounds. This impacts the number of trees created and, thus, decisions of the model and the SHAP analysis. We applied custom parameters to enhance the performance of the model. In the default configuration the model only performs ten rounds of boosting. We have set the maximum number of boosting rounds to 200 to improve the performance. Furthermore, we have set the parameter early_stopping_rounds=5 for efficiency. This parameter stops the training if 5 extra rounds of boosting yield no increased training performance. Using these settings, we see an increase of 2% AUC while the added runtime is negligible.

6.3.1. NCSC-NL

As mentioned before, SHAP allows to identify and analyse TP, FP and FN cases in an efficient manner. In this section we apply SHAP to analyse how the XGBoost model behaves. SHAP analysis shows which features have the most impact on the outcome value. The higher a feature is listed within the summary plot below the more impact it has in the model. So the feature most important for the Dutch NCSC is the number of references. Then for each feature, we can see how the value of the feature corresponds to the SHAP value given. A higher SHAP value for a feature results in an advisory shifting more towards being published whereas a negative value pushes it to being predicted as non-published.

The SHAP value is shown on the X-axis of summary plot while the color coding shows the feature of the value. Whenever more entries have the same SHAP value for a feature, the summary plot will start plotting the points more vertically. All this behaviour can be seen in the plot below. We see that having more references, usually means a higher SHAP value. Furthermore, we see that most entries contain a SHAP value of -1.2 and also have a lower number of references.

SHAP calculates the average SHAP value for all advisories, which is the starting value for each advisory. Then for each feature the value increases or decreases depending on the SHAP value for that feature. If the final value exceeds a threshold selected by SHAP it is predicted as published. This behaviour is shown in SHAP force plots which we shown later on.

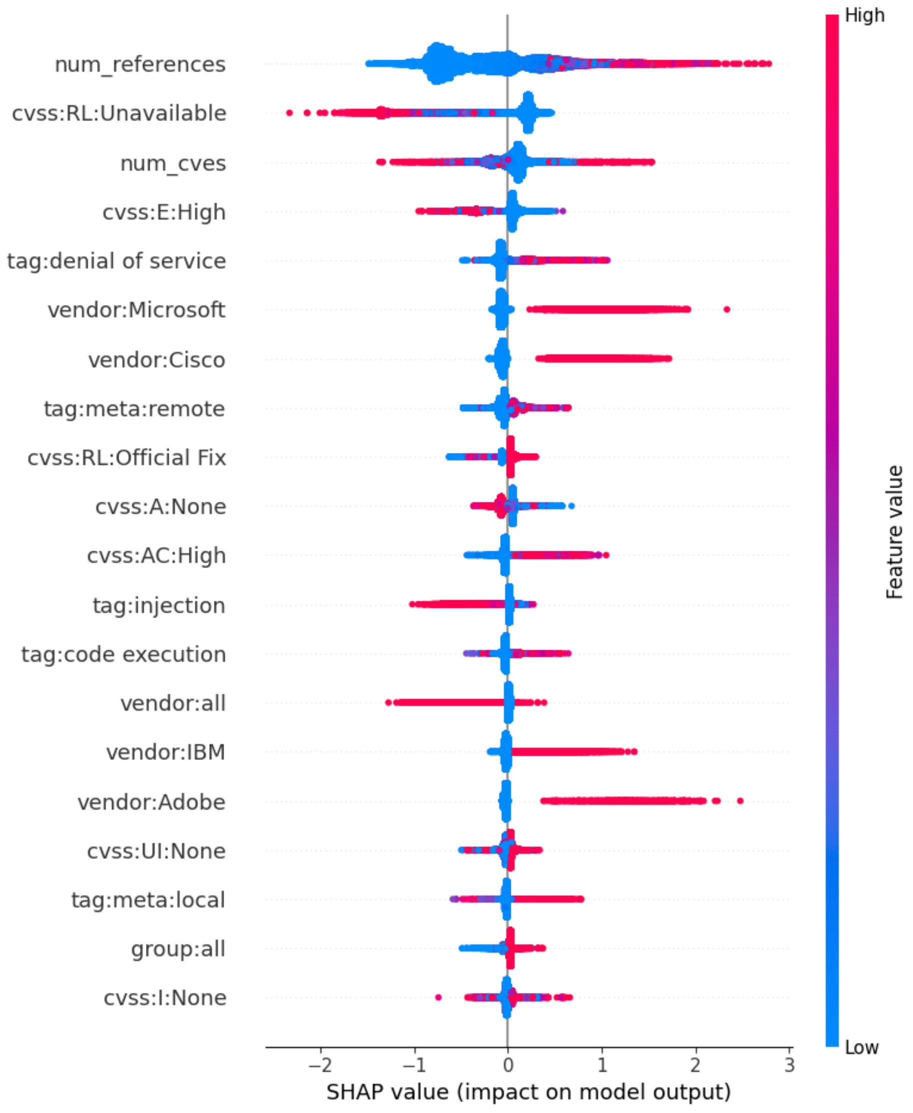


Figure 6.8: SHAP summary plot

True positive
Using the SHAP summary plot above, we can identify cases which are labelled as published. For instance, any advisory which has many CVEs, the tag denial of service and published for Microsoft has a high chance of being marked as published as these value feature combinations have a high positive SHAP value. We identify a true positive by searching for advisories which contain values for features with a high SHAP value. An example of a true positive identified in this way is:

Feature	Value
CVEs	[CVE-2020-36184, CVE-2020-36182, CVE-2020-36183, CVE-2020-35728, CVE-2020-36185, CVE-2020-36188, CVE-2020-36186, CVE-2020-36179, CVE-2020-36187, CVE-2020-36189, CVE-2020-36180, CVE-2020-36181]
Vendor	FasterXML
Product	jackson-databind
Num_references	144.0
CVSS:RL	[Official Fix, Official Fix, Official Fix, Official Fix, Official Fix, Official Fix, Official Fix, Official Fix, Official Fix, Official Fix, Official Fix, Official Fix]
CVSS:RL:Official Fix	1.0
CVSS:RL:Unavailable	0.0
CVSS:E:High	0.0

Table 6.3: Generated advisory

This advisory contains high SHAP values for Num_references, CVSS:RL:Official Fix, CVSS:RL:Unavailable and CVSS:E:High, which results in the advisory being marked as published. This behaviour is also seen in the SHAP force plot below, which we will use in the following case studies to show why an advisory is marked as published or not. To verify it is a true positive case, we see if an similar advisory is published by the Dutch NCSC. The advisory NCSC-2021-0012 is almost identical to the advisory generated. The advisory we generated contains all the same CVEs and a single extra CVE with the ID CVE-2020-35728. This CVE was included as we provide a time window of several days to combine CVE IDs into a single advisory. Hence our matching algorithm found a score of: 11/12 * 100% = 91.7%. This true positive case illustrates why a threshold lower than 1 might be preferable as otherwise this case would not have been identified.

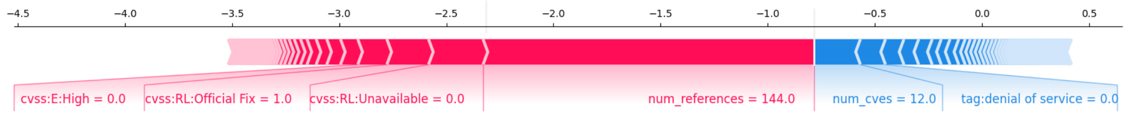


Figure 6.9: SHAP force plot true positive

False positive

Next, we identify a false positive as shown in table 6.4. The advisory is labelled as published by our machine learning model mostly due to the number of references and the presence of the tags bypass and denial of service. However, we see that none of the vulnerabilities within the generated advisory are published by the Dutch NCSC. Therefore, the score of advisory is zero and, thus, is a false positive. The advisory is barely marked as published as can be seen in fig. 6.10. The SHAP value was -1.1 whereas the threshold selected by SHAP was -1.5. Hence, a small adjustment in threshold could result in this false positive being filtered out but this also reduces the number of true positives. Another option would be to reduce the imbalance of advisories published with a large number of CVEs or references. Currently, the vast majority of advisories contain more than 8 vulnerabilities and 20 references. Hence, our trained model quickly marks any advisory with many CVEs or references as published which is not always the case.

Feature	Value
CVEs	[CVE-2020-12951, CVE-2020-12983, CVE-2020-12891, CVE-2020-12986, CVE-2020-12987, CVE-2020-12980, CVE-2020-12954, CVE-2020-12944, CVE-2020-12982, CVE-2020-12964, CVE-2020-12985, CVE-2020-12981]
Vendor	AMD
Product	all
Num_references	24.0

Table 6.4: Generated advisory

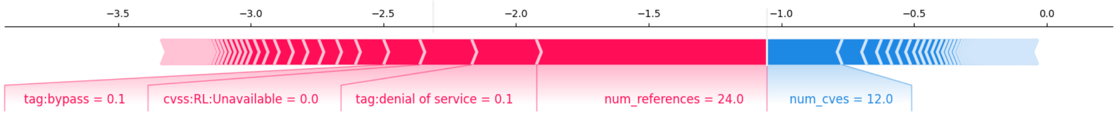


Figure 6.10: SHAP force plot false positive

False negative
The bias we see towards many CVEs is also the reason that there are several false negatives. An example of such a false negative is the generated advisory containing CVE IDs: CVE-2018-10950, CVE-2018-10949, CVE-2018-10951 and CVE-2018-10939. As this advisory only contains few CVE IDs it is less likely to be marked as published. Furthermore, it contains a low value for the exploit status while it has a high value for the availability value None. This results in the advisory receiving a low SHAP value and, thus not being marked as published while it is published under ID NCSC-2018-0550.

The bias towards more vulnerabilities and references is something that cannot easily be removed. We tried to oversample the published advisories with less vulnerabilities but this only decreased the overall performance. Other solutions would be to train separate models depending on the number of vulnerabilities or adjust the number of references feature. We could adjust the number of references to be the average rather than the total number of references. However, we tested this and it decreased the performance. Still, this could be subject for a follow up study.

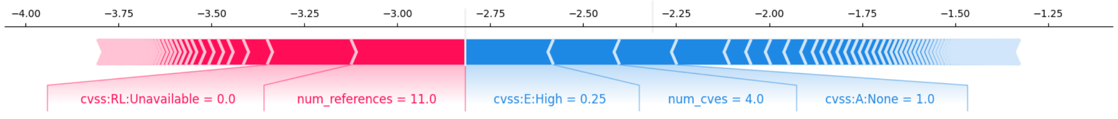


Figure 6.11: SHAP force plot false negative

6.3.2. NCSC-CAN

We repeat our analysis for the Canadian NCSC. In fig. 6.12, we see that the model trained differs a lot from the Dutch model. The number of references and vulnerabilities are less important and instead the so called CIA triad features are the most important: CVSS:C, CVSS:I, CVSS:A.

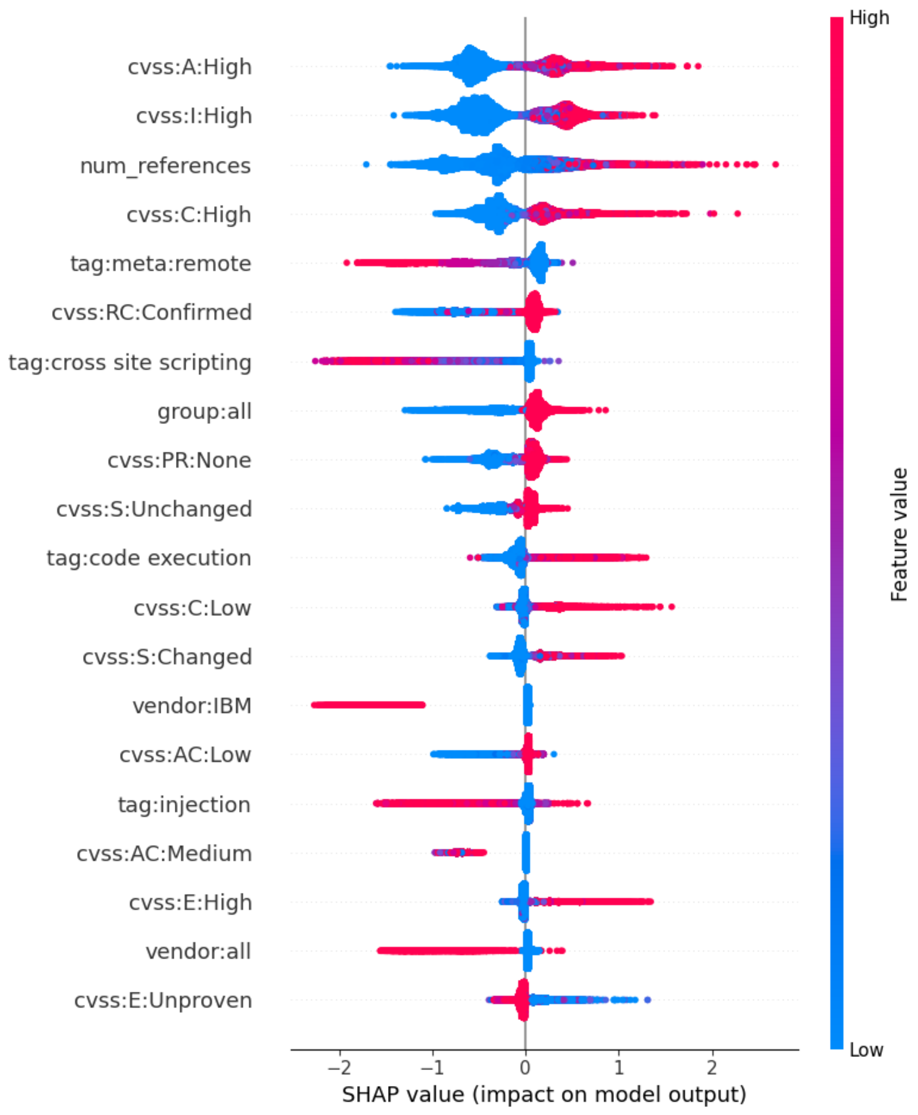


Figure 6.12: SHAP summary plot

True positive

For the Canadian NCSC, we cover two examples of true positives to cover both simple and advanced advisories. For simple advisories, we take an advisory covering CVE IDs: CVE-2019-13551, CVE-2019-18227, CVE-2019-18229 and CVE-2019-13547 [41]. We created the exact same advisory by crawling the URL present in the advisory. Next our machine learning model labelled it as published due to the high values for the CIA triad, many references and the vendor being Advantech. The effect of the values on the SHAP value can be seen in the force plot in fig. 6.13.

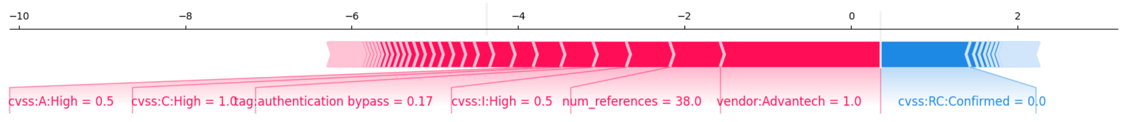


Figure 6.13: SHAP force plot true positive simple

Next is the advanced advisory that belongs to CVE-2021-36958 [42]. Like the simple advisory, it has high values for all CIA triad features. Furthermore, vendor:Microsoft and CVSS:E results in the advisory being marked as published. The way each feature impacts the final SHAP value can be seen below.

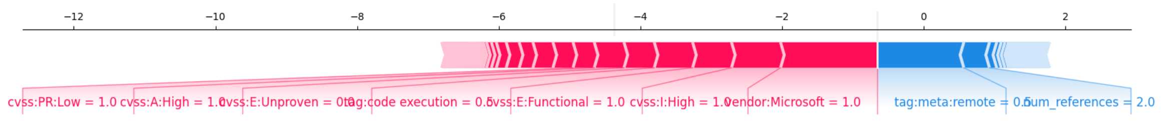


Figure 6.14: SHAP force plot true positive advanced

False positive
The importance of the CIA triad as shown in fig. 6.12 also results in false positives. An example of such is the generated advisory containing: CVE-2017-11073, CVE-2017-11024, CVE-2017-11058, CVE-2017-15398, CVE-2017-11091, CVE-2017-9701, CVE-2017-11085, CVE-2017-9721, CVE-2017-9702, CVE-2017-11092, CVE-2017-9719, CVE-2017-11022, CVE-2017-11090, CVE-2017-11093, CVE-2017-11089, CVE-2017-15399, CVE-2017-8279, CVE-2017-9696, CVE-2017-9690 and CVE-2017-11023. This advisory also has high values for the CIA. But as seen in the force plot the main values which cause it to be published is the tag crafted html which only has a small value. Tags also results in false positives for the Dutch NCSC, this shows that we may need to restructure the tags. We could adjust it to a binary value whether more than X of the vulnerabilities within the advisory contain the tag. This could reduce the chance that whenever a tag is present in just one of the many vulnerabilities, the advisory is not directly marked as published.

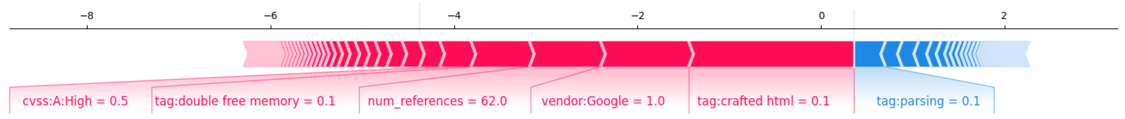


Figure 6.15: SHAP force plot false positive

False negative
Finally, we cover the false negative case for the Canadian NCSC. The advisory regarding vulnerability CVE-2020-10633 is marked as non published as it contains low SHAP values for the most important features (CIA triad and number of references). Furthermore, it is for a rather small vendor which is the reason that there are not a lot of similar vulnerabilities. This imbalance was also the reason for the false negative case for the Dutch NCSC and can be studied further to improve the results.

6.4. Features time variances

Some of the features that we have collected change over time. This may have a negative impact on the performance of the model, since, decision may be taken on incorrect data. Therefore, when we try build a model that predicts whether an advisory is published, we should try to recreate the conditions under which that decision was made. Consider the scenario in which a tag is added to an advisory *after* the decision was made to publish the advisory. In this scenario, the tag did not affect the original decision while a machine learning model might conclude that there is a relation between the tag and an advisory being published. Hence, including time variant features like tags results in more false positives. On the other hand, the usage of time variant features could actually improve the model. The features still represent valuable information and when the change overtime is small, the value added to the model outweighs the inconsistency it adds. We study this behaviour by training three versions of the model:

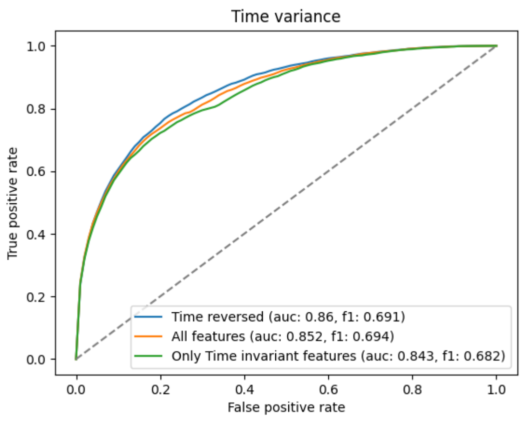
1. All features
2. Only time-invariant features
3. Time-invariant features and all time reversed features

In table 6.5, we show which of the features are time variant and whether it is possible to reverse the time variance. Altering features back to the state during publication is a difficult task and not always possible because some sources have incomplete or no version control. For this reason, we also train the second version of the model. In theory, the performance of the three different versions of the model confirms to what extend the time variance was reversed correctly. We expect the reversed time version to have an increased performance with regard to the other two versions if done correctly. Whereas it should perform similarly or even worse when information is lacking or incorrect.

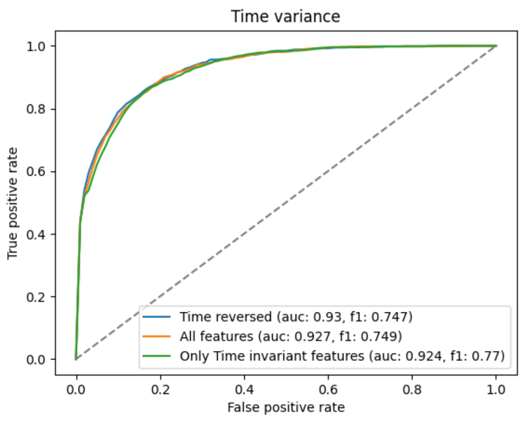
Metric	**Number of features**	**source**	**Time variant**	**Time reverse**
Base score	3	NVD	no	
Base vector	24	NVD	no	
Temporal score	3	X-Force	yes	no
Temporal vector	9	X-Force	yes	no
Weaknesses	25	NVD & MITRE	yes	yes
Categories	100	NVD & MITRE	yes	yes
Reference tags	175	EPSS	yes	no
Github	1	GitHub	yes	no
ExploitDB	1	EDB	yes	no
Metasploit	1	Metasploit	yes	no
Number of References	1	NVD	yes	yes
Number of CVEs	1	Own dataset	no	
Vendor	150	Own dataset	no	
Product	50	Own dataset	no	
Total	544			
Total Time invariant	228			

Table 6.5: Features

Next, we build the three different feature sets and train the XGBoost classifier. The results of the different versions are plotted in fig. 6.16. From the graphs, we see that version with all features (model 1) outperforms the one with just the time invariant features (model 2). This shows that the value added by the features outweighs the inconsistency they contain. However, for both NCSCs the time reverse version (model 3) performs best. This is a promising result as some of the version control was incomplete. In the future, if the model takes snapshots of the data the performance increase could be even higher. The usage of time reverse features does require the timestamp for each published advisory but this easy to implement.



(a) NCSC NL



(b) NCSC Canada

Figure 6.16: Classifiers time variance

7 Discussion

7.1. Findings

The results presented in chapter 5 and chapter 6 are promising. We were able to create a dataset containing almost 90.000 advisories. These advisories are generated by combining vulnerabilities in three different ways:

- Grouping by vendor and publication date
- Grouping by vendor, product type and publication date
- Grouping by NVD reference

This dataset contains identical copies of 82% of the Dutch NCSC advisories and 61% of the Canadian NCSC advisories. As we lower the generating threshold from 1.0 to 0.5, we find that the dataset contains matches for more than 92% of the advisories published by the two organisations. We studied the advisories that did not have a successful match in the dataset. These advisories typically contained human errors or were created using sources not present in the NVD references.

First, the overall performance of both models shows that the dataset created by ACSA is well suited for the automation of cyber security advisories. In the most simple scenario we are able to match around 50% of all advisories that should be published,. i.e. true positives. The number of false positives is limited to a few hundred. In the advanced scenario using the feedback loop a higher percentage can be achieved. The analysts can filtered out false positive and find false negatives. This increase in performance comes at the expense of more manual work by the analysts. This trade-off is a decision that organisations need to take.

The machine learning model trained could also be used for **prioritizing** advisories rather than directly automating them. In this way, ACSA becomes an advice rather than direct automation. The prediction score will then indicate which advisories are the most important for analysts, which could make the manual work more efficient. The ranking could also be displayed to clients, to stress the importance of certain published advisories.

The performance of the model depends on the consistency of deciding which advisories are to be published. For example, if an organisation publishes every advisory which has a CVSS score above nine, with no exceptions, then the decision making is simple and consistent. In that case our machine learning model should achieve much higher scores. In daily practice, however, the decision making process is much more complicated.

Similarly, we can achieve much better results when applying the model to a selection of vendors, rather than all vendors. The reason for this s that the smaller vendors have to few samples resulting in the machine learning model not being able to model their behaviour correctly. Currently our model could match approximately half of the published advisories, but if we would focus on specific vendors, we are able to automate more than 90% of the advisories.

Finally, we have studied the impact of changes to features over time. In some cases we are able to reverse these changes, From the plots, we find that this is beneficial to the performance of the model. Now that we are aware of this effect, we can easily remedy time variance, We could take daily snapshots of features which change over time. Then for each advisory, we select the snapshot from the day prior to the publication of the advisory. Effectively this means we do no longer have to reverse time features. which improves the performance.

7.2. Limitations

During the course of this project, we have identified several limitations and concerns of our work.

7.2.1. Time variance

In practice, we cannot reverse all time variant features. Some features simply did not have a change log and, hence, we do not know what changes were made and when. Consequently we could not reverse them to the time of publishing the advisory. Since the beginning of 2021, we have collected snapshots of the features and with ACSA we can easily continue to do so. This means that the time variance effects become smaller and smaller over time. Eventually, we can filter out all samples prior to 2021 and, thus, remove the remaining inconsistency due to time variance.

7.2.2. Data extraction

In the data collection phase we have used a regular expression to retrieve all CVE IDs on a page. For the majority of webpages this works correctly. In some cases, however, too many vulnerabilities are ex- tracted. This happens for instance, when there multiple advisories have been listed on the same page or when unrelated CVE IDs are mentioned. One of the advisories in which this occurs is CVE-2020-2034. It contains a reference to [48], in which they also mention another vulnerability CVE-2020-2021. This vulnerability is not related and should not be included in the same advisory. In the current implementation, this vulnerability is included and decreases the performance. This issue could be resolved by using a more advanced approach involving natural language processing.

7.2.3. Scope of application

Our dataset is designed to handle advisories concerning vulnerabilities. However, organisations also publish advisories on other topics. An example of such an advisory is [32]. This advisory describes a specific attack technique rather than a set of vulnerabilities that should be remediated. We see this is also the case for the organisations used in this paper. Although all advisories from the Dutch NCSC contain vulnerabilities, this is not the case for the Canadian NCSC. They published 2,000 advisories of which 1513 advisories are related to vulnerabilities. The remaining portion of advisories concerns other topics like threat actors and attack techniques. Hence, not all advisories can be handled by our current model but it still is able to handle the vast majority of the advisories from both organisations.

7.2.4. Decision-making process

The performance of model depends on the decision-making process regarding the publishing of advisories. Whenever this process is simple and consistent, the model will yield excellent scores. However, when the process becomes too complex or inconsistent the performance drops. We see a clear performance difference between the organisations for which we tested the dataset.

Inconsistent decision making cannot be resolved by our model but should be handled by the organisation itself. It is important to note that the more effort an organisation puts into removing inconsistencies from their advisories, the better the automation will work. Organisations should relabel old advisories if they adjust which advisories will be published. Similarly, they should remove human errors from old advisories.

8 Conclusion

This chapter concludes our work on the automation of cyber security advisories. In this chapter, we will rephrase our answer to the research question and provide our final conclusions. After that, we provide an overview of the scientific contributions of this work. Finally, we share our ideas for future work that can be done within the field of automating cyber security advisories.

8.1. Research question

All the work presented in this paper was done in an effort to provide an answer to our research question:

RQ: How can we create a dataset that allows for automating cyber security advisories?

After extensively analysing published advisories, we designed an innovative solution to automate cyber security advisories. By combining vulnerabilities in ways that mimic the behaviour of analysts we are able to create a large dataset containing almost 90,000 advisories. Then using a machine learning model, the dataset is filtered to the subset of advisories that should be published by an organisation. This way, we have achieved automation of cyber security advisories.

From the results, we find that this method does in fact work and that we are able to automate around 50% of the advisories published by both organisations. This constitutes a vast reduction of manual work. The percentage will increases when focusing on specific vendors and when we reduce inconsistencies due to the features that change over time.

8.2. Scientific contributions

This work covers a novel field of research. As described in chapter 3, almost no work is done related to the automation of cyber security advisories for third party publishers. The scientific contributions in this work can be summarised as follows:

1. Creation of a dataset that can be used by organisations and researchers to automate advisories
2. Extensive analysis of the performance of the dataset for two organisations. The analysis shows that the dataset can be used for automating cyber security advisories especially when using it on a selection of vendors.
3. We studied the impact of time variance on classifying whether advisories should be published. The results show that reversing the time variance improves the ROC AUC by around 0.5%. This equates to having a few additional true positives while not introducing additional false positives.

8.3. Future work

In this section we propose any ideas for further research on this topic. We can make improvement regarding the data quality, adding features and validation of the model.

8.3.1. Improving data quality

Further research could focus on the data collection process, which could easily be made more robust. Any improvement here benefits the quality of the dataset immediately.

We could also address the issue of data integrity. Theoretically, malicious entities could insert entries into our dataset by injecting URLs in NVD or altering webpages before we collect them. These actions can lower the performance of the dataset. The data collection algorithm could be enhanced such that suspect URLs are excluded, minimizing the risk of data corruption

8.3.2. Enhancing the machine learning model

We identified many human errors in advisories published. We did not correct those mistakes in our research but this likely improves the scores of the machine learning model. It would be interesting to see if these human errors could be identified and corrected automatically

We can also add more features to improve the machine learning, for instance a feature which indicates whether an exploit-in-the-wild was observed. One can gather this data from EPSS, which have a list of which vulnerabilities have been observed to be exploited. Another option would be to use a local detection network to build our own set of vulnerabilities being exploited. For the Dutch NCSC, we could use the National Detection Network (NDN) present in the Netherlands. In this platform, Dutch threats are shared like vulnerabilities being exploited in the dutch market. This is valuable information for organisations to prioritize advisories and, hence, it likely has a strong correlation to the target feature.

8.3.3. Validating the model

We can apply the model to more advisory publishers. This allows us to verify if the dataset works equally well for other organizations and it will help to identify possible improvements. Currently the dataset is only tested for the Dutch and Canadian NCSCs, but it can be used for any organisation publishing cyber security advisories. Examples of organisations which could be used to validate the model are Rapid7 [51], Fox-IT blogs[20] and the Australian NCSC [7].

Bibliography

[1] Luca Allodi and Fabio Massacci. Comparing vulnerability severity and exploits using case-control studies. *ACM Trans. Inf. Syst. Secur.*, 17(1), aug 2014. ISSN 1094-9224. doi: 10.1145/2630069. URL https://doi-org.tudelft.idm.oclc.org/10.1145/2630069.

[2] Luca Allodi, Marco Cremonini, Fabio Massacci, and Woohyun Shim. The effect of security education and expertise on security assessments: The case of software vulnerabilities. *arXiv preprint arXiv:1808.06547*, 2018.

[3] Luca Allodi, Fabio Massacci, and Julian Williams. The work-averse cyberattacker model: Theory and evidence from two million attack signatures. *Risk Analysis*, 2021.

[4] Mohammed Almukaynizi, Eric Nunes, Krishna Dharaiya, Manoj Senguttuvan, Jana Shakarian, and Paulo Shakarian. Proactive identification of exploits in the wild through vulnerability mentions online. In *2017 International Conference on Cyber Conflict (CyCon US)*, pages 82–88. IEEE, 2017.

[5] Ashish Arora, Rahul Telang, and Hao Xu. Optimal policy for software vulnerability disclosure. *Management Science*, 54(4):642–656, 2008.

[6] Terrence August, Duy Dao, and Kihoon Kim. Market segmentation and software security: Pricing patching rights. *Management Science*, 65(10):4575–4597, 2019.

[7] Australian NCSC. Security advisories, 2022. URL https://www.cyber.gov.au/acsc/view-all-content/advisories?title_op=word&title=&body_value_op=word&body_value=&sort_by=field_date_user_updated_value&sort_order=DESC.

[8] John O Awoyemi, Adebayo O Adetunmbi, and Samuel A Oluwadare. Credit card fraud detection using machine learning techniques: A comparative analysis. In *2017 international conference on computing networking and informatics (ICCNI)*, pages 1–9. IEEE, 2017.

[9] Steve Beattie, Seth Arnold, Crispin Cowan, Perry Wagle, Chris Wright, and Adam Shostack. Timing the application of security patches for optimal uptime. In *LISA*, volume 2, pages 233–242, 2002.

[10] Francois Boechat, Gabriel Ribas, Lucas Senos, Miguel Bicudo, Mateus Schulz Nogueira, Leandro Pfleger de Aguiar, and Daniel Sadoc Menasche. Is vulnerability report confidence redundant? pitfalls using temporal risk scores. *IEEE Security & Privacy*, 19(4):44–53, 2021.

[11] Mehran Bozorgi, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Beyond heuristics: learning to classify vulnerabilities and predict exploits. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 105–114, 2010.

[12] Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications surveys & tutorials*, 18(2):1153–1176, 2015.

[13] Hasan Cavusoglu, Huseyin Cavusoglu, and Srinivasan Raghunathan. Efficiency of vulnerability disclosure mechanisms to disseminate vulnerability knowledge. *IEEE Transactions on Software Engineering*, 33(3):171–185, 2007.

[14] Hasan Cavusoglu, Huseyin Cavusoglu, and Jun Zhang. Security patch management: Share the burden or share the damage? *Management Science*, 54(4):657–670, 2008.

[15] Debabrata Dey, Atanu Lahiri, and Guoying Zhang. Optimal policies for security patch management. *INFORMS Journal on Computing*, 27(3):462–477, 2015.

[16] Michel Edkrantz and Alan Said. Predicting cyber vulnerability exploits with machine learning. In *SCAI*, pages 48–57, 2015.

[17] Daniel S Fava, Stephen R Byers, and Shanchieh Jay Yang. Projecting cyberattacks through variable-length markov models. *IEEE Transactions on Information Forensics and Security*, 3(3): 359–369, 2008.

[18] FIRST. Common vulnerability scoring system version 3.1: Specification document, 2022. URL `https://www.first.org/cvss/specification-document`.

[19] First. Epss v2, 2022. URL `https://www.first.org/epss/model`.

[20] Fox-IT. Fox-it blog, 2022. URL `https://blog.fox-it.com/`.

[21] Frank Fransen, Andre Smulders, and Richard Kerkdijk. Cyber security information exchange to gain insight into the effects of cyber threats and incidents. *e & i Elektrotechnik und Informationstechnik*, 132(2):106–112, 2015.

[22] Palash Goyal, KSM Hossain, Ashok Deb, Nazgol Tavabi, Nathan Bartley, Andr'es Abeliuk, Emilio Ferrara, and Kristina Lerman. Discovering signals from web sources to predict cyber attacks. *arXiv preprint arXiv:1806.03342*, 2018.

[23] Marti A Hearst. Automatic acquisition of hyponyms from large text corpora. In *COLING 1992 Volume 2: The 14th International Conference on Computational Linguistics*, 1992.

[24] Pilar Holgado, Víctor A Villagrá, and Luis Vazquez. Real-time multistep attack prediction based on hidden markov models. *IEEE Transactions on Dependable and Secure Computing*, 17(1): 134–147, 2017.

[25] Allen D Householder, Jeff Chrabaszcz, Trent Novelly, David Warren, and Jonathan M Spring. Historical analysis of exploit availability timelines. In *13th {USENIX} Workshop on Cyber Security Experimentation and Test ({CSET} 20)*, 2020.

[26] IBM. X-force exchange, 2022. URL `https://exchange.xforce.ibmcloud.com/`.

[27] Jay Jacobs, Sasha Romanosky, Benjamin Edwards, Michael Roytman, and Idris Adjerid. Exploit prediction scoring system (epss), 2019. URL `https://arxiv.org/abs/1908.04856`.

[28] Jay Jacobs, Sasha Romanosky, Idris Adjerid, and Wade Baker. Improving vulnerability remediation through better exploit prediction. *Journal of Cybersecurity*, 6(1), 09 2020. ISSN 2057-2085. doi: 10.1093/cybsec/tyaa015. URL `https://doi.org/10.1093/cybsec/tyaa015`. tyaa015.

[29] Jay Jacobs, Sasha Romanosky, Benjamin Edwards, Idris Adjerid, and Michael Roytman. Exploit prediction scoring system (epss). *Digital Threats: Research and Practice*, 2(3):1–17, 2021.

[30] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[31] Arnav Joshi, Ravendar Lal, Tim Finin, and Anupam Joshi. Extracting cybersecurity related linked data from text. In *2013 IEEE Seventh International Conference on Semantic Computing*, pages 252–259, 2013. doi: 10.1109/ICSC.2013.50.

[32] Arnav Joshi, Ravendar Lal, Tim Finin, and Anupam Joshi. Extracting cybersecurity related linked data from text. In *2013 IEEE Seventh International Conference on Semantic Computing*, pages 252–259. IEEE, 2013.

[33] Kenna Security. Winning the remediation race, 2019. URL `https://www.kennasecurity.com/resources/prioritization-to-prediction-report-volume-three/`.

[34] Konstantina Kourou, Themis P Exarchos, Konstantinos P Exarchos, Michalis V Karamouzis, and Dimitrios I Fotiadis. Machine learning applications in cancer prognosis and prediction. *Computational and structural biotechnology journal*, 13:8–17, 2015.

[35] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pages 404–411, 2004.

[36] MITRE. 2021 cwe top 25 most dangerous software weaknesses, 2022. URL `https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html`.

[37] MITRE. Common vulnerabilities and exposures, 2022. URL `https://www.cve.org/`.

[38] Varish Mulwad, Wenjia Li, Anupam Joshi, Tim Finin, and Krishnamurthy Viswanathan. Extracting information about security vulnerabilities from web text. In *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, volume 3, pages 257–260. IEEE, 2011.

[39] Saurav Nanda, Faheem Zafari, Casimer DeCusatis, Eric Wedaa, and Baijian Yang. Predicting network attack patterns in sdn using machine learning approach. In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 167–172. IEEE, 2016.

[40] National Institure of Standards and Technology. Exploit prediction scoring system (epss), 2022. URL `https://nvd.nist.gov/general/visualizations/vulnerability-visualizations/cvss-severity-distribution-over-time`.

[41] NCSC-CAN. Advantech security advisory, 2019. URL `https://cyber.gc.ca/en/alerts-advisories/control-systems-advantech-security-advisory-0`.

[42] NCSC-CAN. Alert - ongoing vulnerabilities involving windows print spooler, 2021. URL `https://cyber.gc.ca/en/alerts-advisories/ongoing-vulnerabilities-involving-windows-print-spooler`.

[43] NCSC-CAN. Cyber advisory, 2022. URL `https://cyber.gc.ca/en/alerts/spring-remote-code-execution-vulnerabilities`.

[44] NCSC-CAN. Canadian centre for cyber security, 2022. URL `https://cyber.gc.ca/en`.

[45] NCSC Canada. Microsoft edge security advisory (av22-238), April 2022. URL `https://cyber.gc.ca/en/alerts/microsoft-edge-security-advisory-av22-238`.

[46] NVD. Api vulnerabilities, 2022. URL `https://nvd.nist.gov/developers/vulnerabilities`.

[47] Ahmet Okutan, Shanchieh Jay Yang, and Katie McConky. Predicting cyber attacks with bayesian networks using unconventional signals. In *Proceedings of the 12th Annual Conference on Cyber and Information Security Research*, pages 1–4, 2017.

[48] PaloAlto Networks. Cve-2020-2034 pan-os: Os command injection vulnerability in globalprotect portal, 2020. URL `https://security.paloaltonetworks.com/CVE-2020-2034r`.

[49] Roshni R. Ramnani, Karthik Shivaram, Shubhashis Sengupta, and Annervaz K. M. Semi-automated information extraction from unstructured threat advisories. In *Proceedings of the 10th Innovations in Software Engineering Conference*, ISEC '17, page 181–187, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450348560. doi: 10.1145/3021460.3021482. URL `https://doi.org/10.1145/3021460.3021482`.

[50] Konstantinos Rantos, Arnolnt Spyros, Alexandros Papanikolaou, Antonios Kritsas, Christos Ilioudis, and Vasilios Katos. Interoperability challenges in the cybersecurity information sharing ecosystem. *Computers*, 9(1), 2020. ISSN 2073-431X. doi: 10.3390/computers9010018. URL `https://www.mdpi.com/2073-431X/9/1/18`.

[51] Rapid7. Vulnerability & exploit database, 2022. URL `https://www.rapid7.com/db/`.

[52] Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. Automatic keyword extraction from individual documents. *Text mining: applications and theory*, 1:1–20, 2010.

[53] Carl Sabottke, Octavian Suciu, and Tudor Dumitra□. Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 1041–1056, 2015.

[54] Avi Shaked and Oded Margalit. Ontorisk – a formal ontology approach to automate cyber security risk identification. In *2022 17th Annual System of Systems Engineering Conference (SOSE)*, pages 74–79, 2022. doi: 10.1109/SOSE55472.2022.9812653.

[55] Jonathan Spring, Eric Hatleback, A Manion, and D Shic. Towards improving cvss. *SEI, CMU, Tech. Rep*, 2018.

[56] Jonathan Spring, Eric Hatleback, Allen Householder, Art Manion, and Deana Shick. Time to change the cvss? *IEEE Security & Privacy*, 19(2):74–78, 2021.

[57] Jonathan M Spring, Eric Hatleback, Allen Householder, Art Manion, and Deana Shick. Prioritizing vulnerability response: A stakeholder specific vulnerability categorization. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA PITTSBURGH United States, 2019.

[58] Ubuntu. Security notices, 2022. URL `https://ubuntu.com/security/notices`.

[59] Chensi Wu, Tao Wen, and Yuqing Zhang. A revised cvss-based system to improve the dispersion of vulnerability risk scores. *Science China Information Sciences*, 62(3):1–3, 2019.

[60] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *33rd annual meeting of the association for computational linguistics*, pages 189–196, 1995.

[61] Wanying Zhao and Gregory White. An evolution roadmap for community cyber security information sharing maturity model. In *Proceedings of the 50th Hawaii international conference on system sciences*, 2017.

A
Features

Metric	Metric values	Numerical values
Attack Vector (AV)	Network (N), Adjacent (A), Local (L), Physical (P)	0.85, 0.62, 0.55, 0.2
Attack Complexity (AC)	Low (L), High (H)	0.77, 0.44
Privileges Required (PR)	None (N), Low (L), High (H)	0.85, 0.62, 0.27
User Interaction (UI)	None (N), Required (R)	0.85, 0.62
Scope (S)	Unchanged (U), Changed (C)	0.56, 0.22, 0
Confidentiality (C)	High (H), Low (L), None (N)	0.56, 0.22, 0
Integrity (I)	High (H), Low (L), None (N)	0.56, 0.22, 0
Availability (A)	High (H), Low (L), None (N)	0.56, 0.22, 0
Exploit Code Maturity (E)	High (H), Functional (F), Proof-of-Concept (P), Unproven (U)	1, 0.97, 0.94, 0.91
Remediation Level (RL)	Unavailable (U), Workaround (W), Temporary Fix (T), Official Fix (O)	1, 0.97, 0.96, 0.95
Report Confidence (RC)	Confirmed (C), Reasonable (R), Unknown (U)	1, 0.96, 0.92

Table A.1: CVSS metric values

B

Decision trees

B.1. NCSC-NL

B.1.1. Final Tree

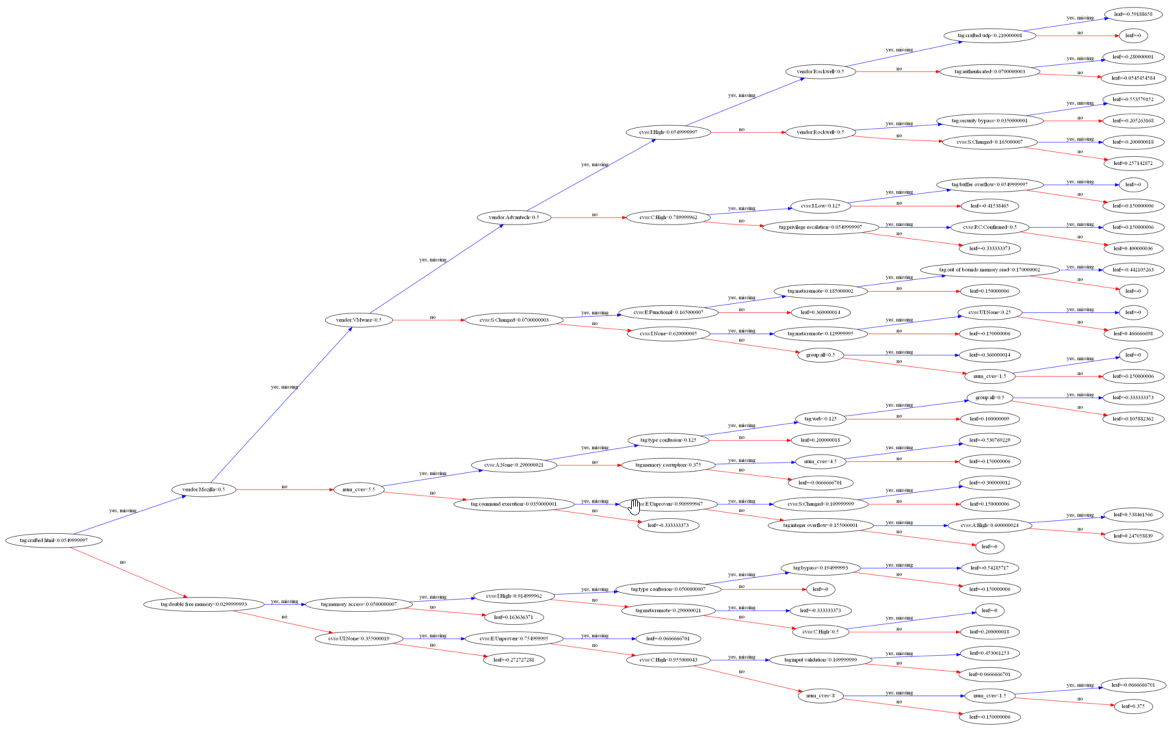


Figure B.1: Decision tree final round

B.2. NCSC-CAN

B.2.1. Final Tree

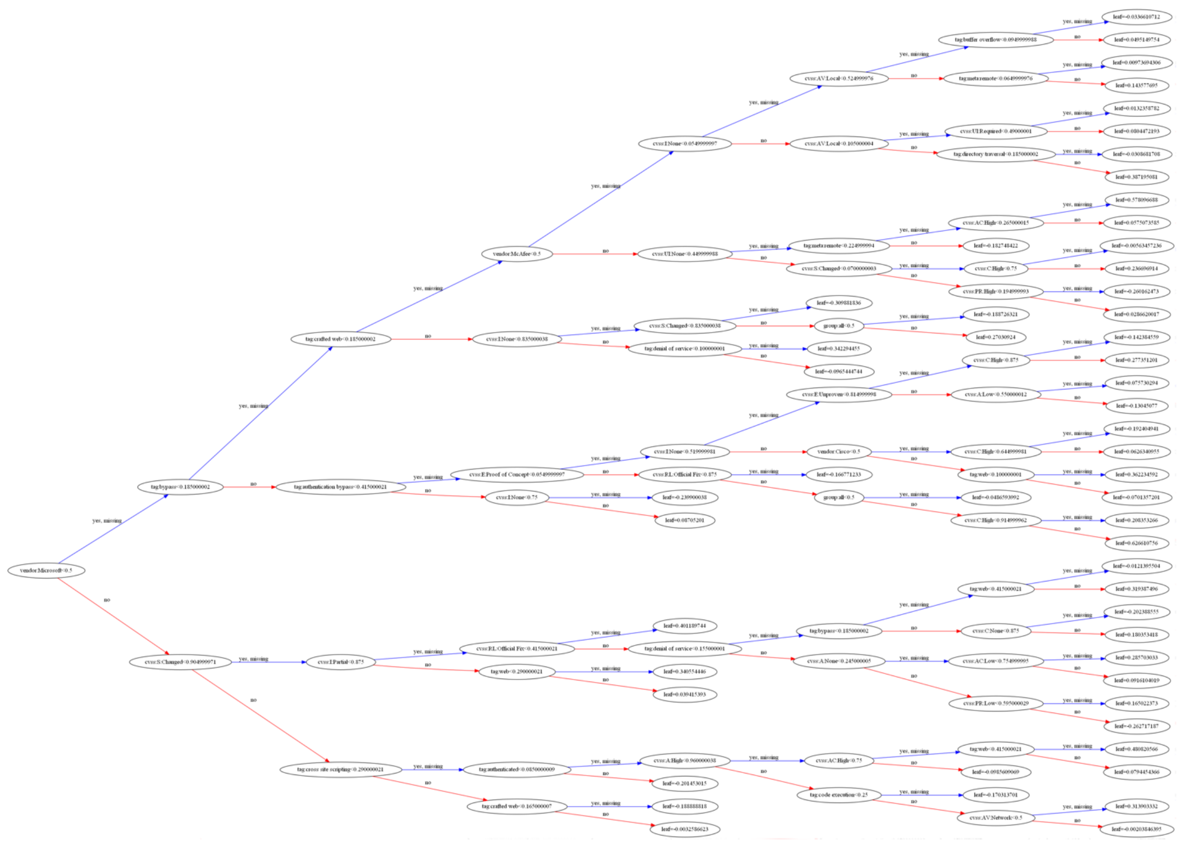
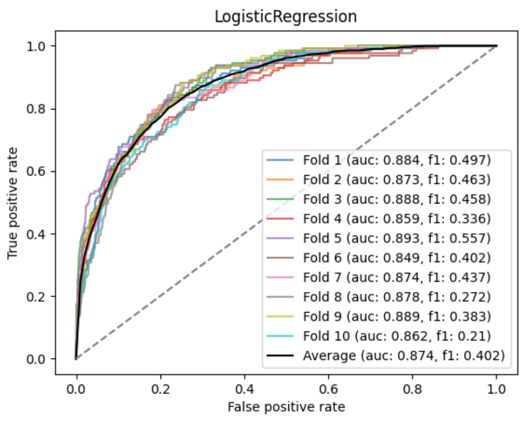
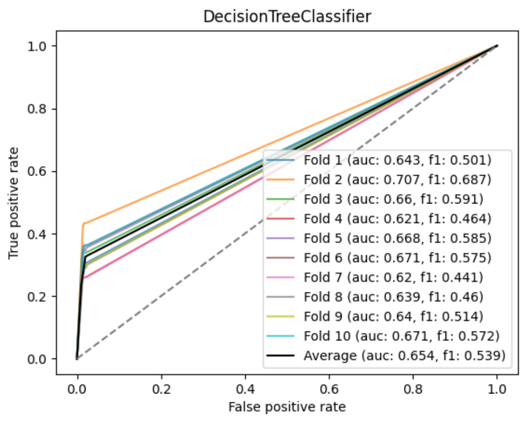


Figure B.2: Decision tree final round

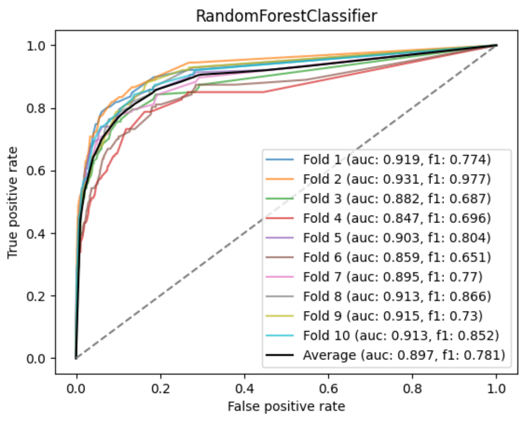
C
Canadian NCSC classifier folds



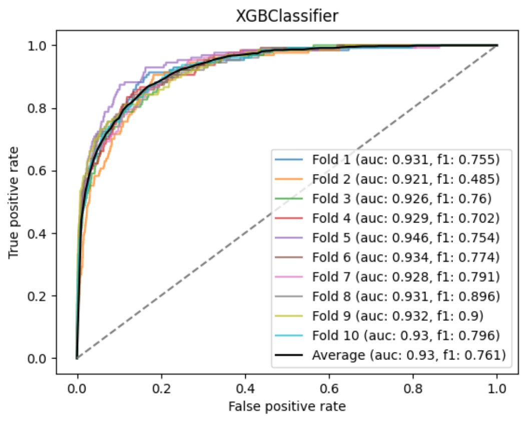
(a) Logistic regression



(b) Decision tree



(a) Random forest classifier



(b) XGBoost classifier

Figure C.2: Random forest & XGBoost classifiers

D Vocabulary

- **IOC (Indicators of Compromise)**, a term used to refer to specific artifacts left by an intrusion, or greater sets of information that allow for the detection of intrusions or other activities conducted by attackers. This may involve mentions of attacks like DOS, DDOS or specific CVE identification numbers.
- **Threat Actors**: specific individuals or groups responsible for the security attack. They tend to include organized cyber criminals and hacktivists.
- **Campaigns**: specific organizations, organization types and countries which are the target of the attack.
- **TTP (Tactics, Techniques and Procedures)** details how an attack could be performed including specific attack patterns, tools and infrastructure used.
- **COA (Course of Action)** details recommendations or guidelines for preventing the attack. They could be as simple as following of recommended guidelines for update of software to detailed actions that must be taken to prevent the attack.
- **Exploit Target** is the software or hardware device which could be affected in the attack.
- **Advisory publisher**, any organisation which uses various sources and vulnerabilities to publish an advisory. The advisories inform their target audience on which vulnerabilities to mitigate. These are organisations like the NCSC-NL and NCSC-CAN who can profit from apply the model demonstrated in this research.
- **Real advisory**, an advisory which has been published by an advisory publisher. Opposed to a generated advisory, which are created in this paper.
- **Generated advisory**, an advisory which has been assembled by combining multiple vulnerabilities. The way we combine these vulnerabilities is covered in chapter 5.