Fighting N-Day Vulnerabilities with Automated CVSS Vector Prediction at Disclosure

Clément Elbaz, Louis Rilling, Christine Morin

Fighting N-Day Vulnerabilities with Automated CVSS Vector Prediction at Disclosure

Clément Elbaz
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
clement.elbaz@inria.fr

Louis Rilling
DGA
Rennes, France
louis.rilling@irisa.fr

Christine Morin
Inria
Rennes, France
christine.morin@inria.fr

ABSTRACT

The Common Vulnerability Scoring System (CVSS) is the industry standard for describing the characteristics of a software vulnerability and measuring its severity. However, during the first days after a vulnerability disclosure, the initial human readable description of the vulnerability is not available as a machine readable CVSS vector yet. This situation creates a period of time when only expensive manual analysis can be used to react to new vulnerabilities because no data is available for cheaper automated analysis yet. We present a new technique based on linear regression to automatically predict the CVSS vector of newly disclosed vulnerabilities using only their human readable descriptions, with a strong emphasis on decision explicability. Our experimental results suggest real world applicability.

KEYWORDS

Security, CVSS, CVE, Machine Learning, Linear Regression

1 INTRODUCTION

Disclosure is the most critical part of a vulnerability life cycle. Indeed, zero-day vulnerabilities are confidential, high value assets used sparingly against high value targets while well known public vulnerabilities can be mitigated using standard security practices: by applying software updates in a diligent manner or using a signature-based intrusion detection system (IDS).

Bilge et al. [17] showed a five orders of magnitude increase in the usage of vulnerabilities before and after their disclosure. In the midst of this increasing threat, most standard defense mechanisms do not work at disclosure: software patches have not been deployed and sometimes are not even available yet. Security experts do not understand the vulnerability well enough to author a proper signature rule for an IDS at this early stage. These factors contribute in making the disclosure a dangerous time, leaving a lot of systems vulnerable in practice.

N-day vulnerabilities are newly disclosed vulnerabilities that are still in the critical part of their life cycle *n* days after their disclosure. While the value of *n* can vary with vulnerabilities and organizations, successfully defending systems against *n-day* vulnerabilities involves keeping the value of *n* as close to 0 as possible.

The process of disclosing new vulnerabilities is coordinated by the *Common Vulnerabilities and Exposures* (CVE) system overseen by Mitre's Corporation [6]. Newly disclosed vulnerabilities are first published on the *CVE List* data feed (managed by Mitre) then forwarded to other security databases such as NIST's *NVD database* [13] or SCAP data feeds [15]. Only then they will be annotated by multiple security experts. These annotations include a threat analysis comprising the redaction of a *Common Vulnerability Scoring System* (CVSS) vector and score [7].

Security experts such as NIST's take days or even weeks to analyze and annotate a vulnerability (see Section 3.1). It is common to find vulnerabilities that have been disclosed for several days that are still not analyzed by NVD. For instance CVE-2020-0583, disclosed on the CVE List on 03/12/2020, has no NVD analysis as of 03/22/2020. This means that it is not possible to rely on enriched metadata provided by databases such as NVD to analyze n-day vulnerabilities. Instead one should focus on the only elements available at disclosure: a unique CVE identifier, a free-form human readable description, and at least one public reference [10].

The current state of this ecosystem makes it expensive for organizations to analyze vulnerabilities at disclosure. Achieving real-time threat evaluation of new vulnerabilities through manual analysis is possible but requires extensive manpower as hundreds of vulnerabilities are disclosed daily. Automated real-time threat evaluation is not currently practical as there is not enough machine-readable metadata available at disclosure for automated analysis. Real-time threat analysis is therefore prohibitively expensive for most organizations even though it would benefit them as severe vulnerabilities such as Shellshock [3] have been massively exploited within hours of their disclosure.

Real-time threat evaluation for newly disclosed vulnerabilities would be more affordable to organizations if it could be automated. It would allow real-time reaction to vulnerability disclosures from cloud service providers (CSP) and information systems. These automated reactions can include reconfiguration of security policies such as logging level elevation for critical systems, switching these systems into degraded mode, or even shutting them down while waiting for a remediation to be applied. Such a reaction service could give the CSP the opportunity to protect both its internal systems and its tenants (the latter constituting a potential source of revenues for the CSP).

We propose an automated system that uses the free-form text descriptions of newly-disclosed vulnerabilities to predict the CVSS base vector of these vulnerabilities, and can do so in near real-time (at most seconds after disclosure) while maintaining a strong explicability chain between the input data and the prediction. To the best of our knowledge this is the first attempt at doing so.

In Section 2 we discuss related work and the real world challenges of working with vulnerability data. In Section 3 we present our approach. In Section 4 we evaluate the accuracy of our proposed technique. We discuss our results in Section 5 and conclude in Section 6.

Base Vector		Temporal Vector	
CVSS V2	CVSS V3	CVSS V2	CVSS V3
Access Vector (AV)	Attack Vector (AV)	Exploitability (E)	Exploit Code Maturity (E)
Access Complexity (AC)	Attack Complexity (AC)	Remediation Level (RL)	Remediation Level (RL)
Authentication (Au)	Privileges Required (PR)	Report Confidence (RC)	Report Confidence (RC)
	User Interaction (UI)	**Environmental Vector**	
Confidentiality Impact (C)	Confidentiality (C)	**CVSS V2**	**CVSS V3**
Integrity Impact (I)	Integrity (I)	Collateral Damage Potential (CDP)	Modified Base Metrics (M*)
Availability Impact (A)	Availability (A)	Target Distribution (TD)	
	Scope (S)	Confidentiality Requirement (CR)	Confidentiality Requirement (CR)
		Integrity Requirement (IR)	Integrity Requirement (IR)
		Availability Requirement (AR)	Availability Requirement (AR)

Table 1: Fields for Base, Temporal, and Environmental CVSS vectors in V2 and V3.

Description			
The HTTP/2 implementation in Apache Tomcat 9.0.0.M1 to 9.0.14 and 8.5.0 to 8.5.37 accepted streams with excessive numbers of SETTINGS frames and also permitted clients to keep streams open without reading/writing request/response data. By keeping streams open for requests that utilised the Servlet API's blocking I/O, clients were able to cause server-side threads to block eventually leading to thread exhaustion and a DoS.			
CVSS V3 Base Vector			
CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H			
CVSS V2 Base Vector			
AV:N/AC:L/Au:N/C:N/I:N/A:P			
CVSS V3 Base Score	7.5	**CVSS V2 Base Score**	5.0

Table 2: Description, CVSS V2 and V3 base vectors and scores for vulnerability CVE-2019-0199.

2 BACKGROUND AND RELATED WORK

CVSS is the de facto standard when formally describing software vulnerabilities properties and severities. Multiple versions of the standard have been released, with CVSS V2 [1], CVSS V3 [8], CVSS V3.1 [9] all being widely used in the computer security community. All versions share the same general architecture: a series of multiple-choice questions to be answered about the vulnerability, separated into three groups: the *base group*, describing the inherent properties of the vulnerability, the *temporal group* describing some of the vulnerability properties that can evolve with time, and the *environmental group*, describing the severity of a vulnerability in the context of a specific organization. Table 1 shows the different fields required by CVSS V2 and V3. When all answers about a vulnerability have been filled, they are combined into a *CVSS Vector* using a standard syntax. Each version of the CVSS specification provides a *severity formula*, taking all fields as input and returns a *severity score* between 0.0 and 10.0. This computation is not linear and small changes in the vector can lead to big differences in the severity score. CVSS V3 and V3.1 share the same fields and only differ by a minor change in the severity formula that only impacts a small number of vulnerabilities. In the rest of our work we use the term "CVSS V3" to describe both the CVSS V3 and V3.1 specifications: when severity calculation is involved, we use the 3.1 variant unless specified. As an example, Table 2 provides CVSS V2 and V3 base vectors and scores for CVE-2019-0199, a vulnerability disclosed in April 2019 affecting Apache Tomcat.

Khazaei et al already explored CVSS severity score prediction [25]: they compared the use of SVM, Random Forests, and fuzzy systems to predict CVSS base scores, in both offline and online environments. However their work differs from ours in a number of ways:

- Their approach is focused on the severity score of the vulnerability, and they do not predict it completely: they approach severity prediction as a classification problem, treating the integer part of the score as a class to be predicted. To the best of our knowledge, our approach is the first to reconstruct a full CVSS vector, thus providing more details about the vulnerability and allowing an actual CVSS score to be recomputed.
- They do not take results explicability into account. Both their approach and ours start by treating a vulnerability description as a bag-of-words (in their case, they apply a TF-IDF [24] weighting scheme to the word count). They use dimension reduction methods (Linear Discriminant Analysis and Principal Component Analysis) that create latent variables that cannot be easily interpreted. Moreover, some of the classifier algorithms they use, such as random forest or fuzzy systems, do not exhibit strong explicability properties either. We consider the explicability of automated analysis as a paramount quality of security systems and our prediction pipeline is designed to preserve explicability at all stages, as described in Section 3.
- Furthermore, their experimental protocol evaluates the prediction as a binary event: either the correct class has been predicted, or not. In our opinion there is a strong difference between incorrectly classifying a vulnerability with an actual severity score of 8.x as 7.y and 2.z. In our evaluation protocol we evaluate the prediction of each vector component individually and study the severity score prediction as a numerical error.
- Both their evaluation protocol and ours include an "online" evaluation, evaluating how a vulnerability can be analyzed

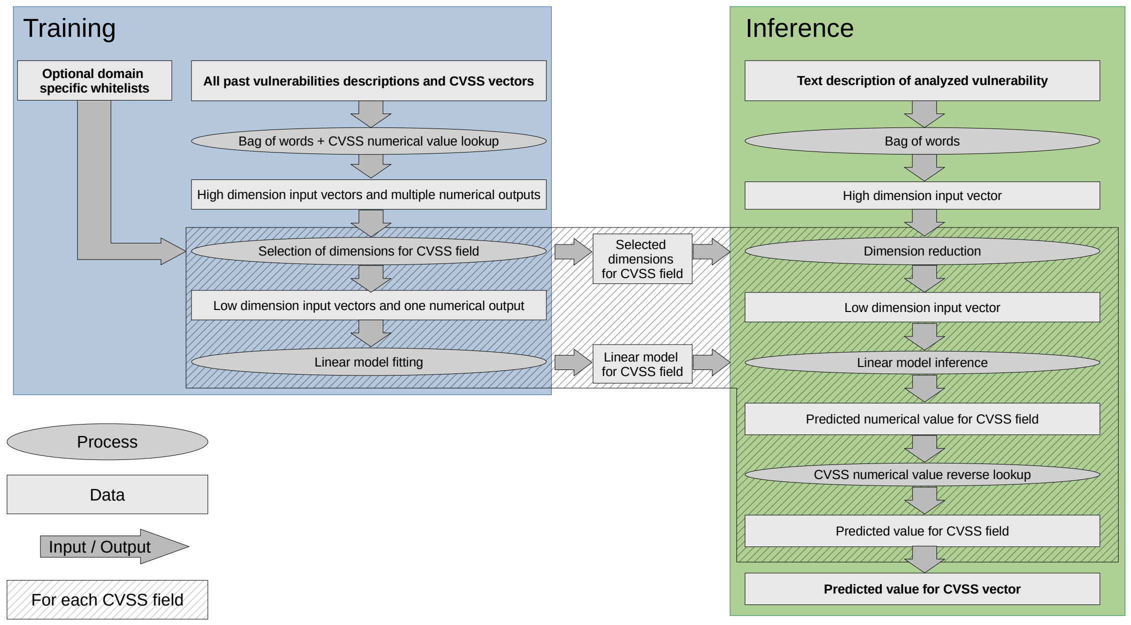

Training
Optional domain specific whitelists
All past vulnerabilities descriptions and CVSS vectors
Bag of words + CVSS numerical value lookup
High dimension input vectors and multiple numerical outputs
Selection of dimensions for CVSS field
Low dimension input vectors and one numerical output
Linear model fitting
Selected dimensions for CVSS field
Linear model for CVSS field
Inference
Text description of analyzed vulnerability
Bag of words
High dimension input vector
Dimension reduction
Low dimension input vector
Linear model inference
Predicted numerical value for CVSS field
CVSS numerical value reverse lookup
Predicted value for CVSS field
Predicted value for CVSS vector
Process
Data
Input / Output
For each CVSS field

Figure 1: Our CVSS vector prediction pipeline.

using the data present at its disclosure. However they do not take into account that CVSS information is not immediately available after disclosure, something we take into account. Moreover, their online evaluation uses monthly steps. This is not granular enough for evaluating a technique robustness for n-day vulnerabilities. Our evaluation protocol uses weekly steps for all variants of our technique and daily steps for the most promising variants.

- As our work is more recent, we evaluate our technique on both CVSS V2 and CVSS V3 using vulnerability data up to 2019 included. Their work only considers CVSS V2 and data up until 2013 included.

Jacobs et al. [23] proposed the Exploit Prediction Scoring System (EPSS). Like our work, EPSS is meant to be used at vulnerability disclosure: they try to determine a vulnerability's probability of exploitation in the twelve months after its disclosure. They use a logistic regression model trained using both public and non-public data sources that can infer probabilities for new vulnerabilities using only public data sources. The two main differences between their work and ours are that they try to predict a different information than we do, and that we do not require any private data source in our pipeline.

Multiple works brought meaningful insights using statistical analyses of historical vulnerabilities in the CVE corpus. Frei et al. [19] found a statistical correlation between the availability of exploits and patches and the number of days since disclosure. Clark et al. [18] brought to light a "honeymoon effect" where more recent software is less subject to new vulnerabilities than older software, everything else being equal. Ganz et al. [20] attempted to automatically enrich the quality of the metadata in NVD, by blending the existing metadata with textual analysis of the description. However their technique still requires the availability of existing metadata for the vulnerability, while ours does not.

Most cloud providers provide Intrusion Prevention System (IPS) or Web Application Firewall (WAF) capabilities among their commercial offering [2][12][5]. However, to the best of our knowledge, the process of monitoring new vulnerabilities and adding related rules is always done manually [4].

3 PROPOSED APPROACH

In this section we describe our CVSS vector prediction pipeline. Its input is the free-form description of a new vulnerability. It is analyzed using all vulnerability descriptions and metadata available at the time of disclosure. It outputs a predicted CVSS base vector (in V2 or V3 format). A high-level overview of the proposed vulnerability analysis pipeline is shown in Figure 1. We now describe each stage of the pipeline in more details, starting with our choice of data sources.

3.1 Data Availability at Prediction Time

We use data from the NVD CVE database, as it includes descriptions (which we use as training *input*) and vulnerability CVSS vectors (which we use as training *output*) for all CVE vulnerabilities. However as we saw in Section 1, metadata (including CVSS vectors) is not published at disclosure time, but authored by security experts several days after. Figure 2 shows the number of days between vulnerability disclosure and analysis publication in NVD from 2007 to 2019. Historically the median analysis duration has been zero day while the 9th decile has been two days. While this remained true until 2016 (for the median) and 2012 (for the 9th decile), there have been sharp drops in NVD analyses timeliness since then. In 2018 the median and 9th decile analysis duration reached 35 days and 63

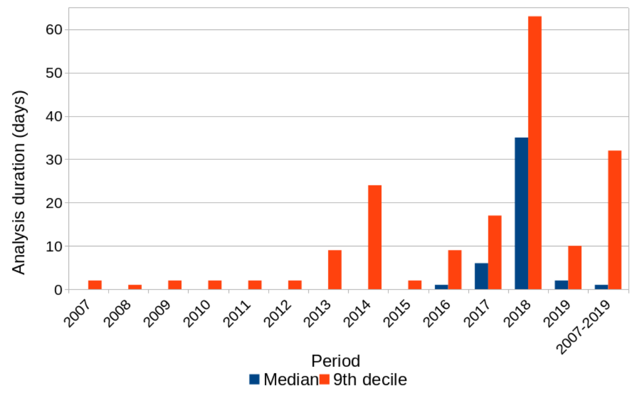


Figure 2: Number of days between vulnerability disclosure and analysis in NVD from 2007 to 2019.

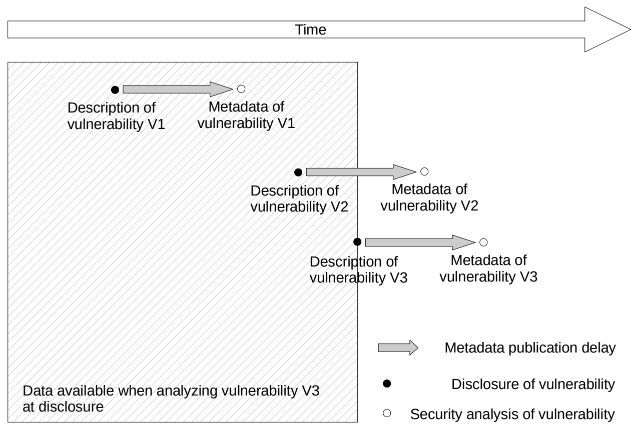

Time
Description of vulnerability V1
Metadata of vulnerability V1
Description of vulnerability V2
Metadata of vulnerability V2
Description of vulnerability V3
Metadata of vulnerability V3
Metadata publication delay
Disclosure of vulnerability
Security analysis of vulnerability
Data available when analyzing vulnerability V3 at disclosure

Figure 3: In this example, when analyzing the text description of vulnerability V3 at disclosure time we have access to the text descriptions for V1 and V2 and to the metadata for V1, but not the metadata for V2 or V3.

days respectively. While NVD should be credited for considerably improving the 2019 delays compared to 2017 and 2018, this shows availability of metadata can not be taken for granted at disclosure. Figure 3 shows a simplified example of how time impacts the data available when analyzing vulnerabilities at disclosure, and we detail in Section 4 how we take metadata publication delay into account in our evaluation protocol. Whitelist-based dimension reduction uses additional data sources which we describe in Section 3.3.1.

3.2 Training Pipeline and Explicability

Beyond accuracy we had one major goal when designing our machine learning pipeline: keeping an explicable and reliable relationship between the input and the decision. This is important for security systems as most organizations require security incidents to be audited and their root cause understood. When such an incident occurs because of an incorrect decision from an inference system, the decision process of the inference system becomes the root cause to be analyzed. If the failure mode of the inference system cannot be understood, the correct course of action to prevent future occurrences of the incident is not to rely on the inference system anymore, often leading to its decommissioning.

For the sake of explicability we also have to limit the number of hyperparameters in the pipeline. A hyperparameter is a parameter that has to be chosen before training occurs, has a direct correlation with model performance, but has a non-obvious impact on model accuracy. The space of all possible combinations of hyperparameters settings for a given training pipeline is called an *hyperstate* and grows exponentially with the number of hyperparameters. The optimal choice of hyperparameter settings is very much dependent on the training and test data, which in online problems such as ours evolve with time. As hyperparameters must be set before the beginning of training (which can be lengthy), a highly dimensional hyperstate cannot be fully explored repeatedly, creating risks of accuracy drifts over time or subtle differences in failure modes when inferring. We would like all variations of our training pipeline to have either zero or one hyperparameter, allowing for a full understanding of the model hyperstate.

To handle these constraints, we propose the pipeline described in Figure 1. We first adopt a *bag of words* approach on each vulnerability description, creating an index of every word ever used in a vulnerability description, then treating every vulnerability as a highly dimensional vector with each dimension being the count of occurrences for a given word (zero indicating its absence from the description text). As the number of words in the index grows loosely linearly with the number of vulnerabilities, we apply a *filtering* scheme (described in Section 3.3) removing irrelevant words in order to manage the number of dimensions of the vulnerability vectors. We then train one regression model (described in Section 3.4) for every component of the CVSS vector using the vulnerability vectors as input. The impact of explicability on accuracy can be quantified through a robust evaluation protocol, as described in Section 4.

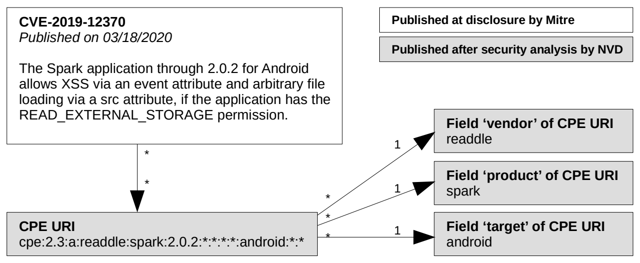
3.3 Dimension Reduction Through Filtering

As we wish to preserve the auditability of the bag-of-word embedding (each dimension counts the number of occurrences of a word) we choose not to use any dimension reduction technique creating latent variables. This excludes techniques such as Linear Discriminant Analysis, Principal Component Analysis, Locality Sensitive Hashing [22], and many others.

Instead, we retain existing dimensions but filter out the ones we deem irrelevant. We propose two approaches for that: domain-specific whitelists, and dimension sorting using conditional entropy.

3.3.1 Domain-specific whitelists. We use two pieces of data as sources of terms for generating a keyword whitelist: Common Platform Enumeration (CPE) URIs [14] and Mitre's Common Weakness Enumeration (CWE) database [11].

A CPE URI is a unique reference to a specific entry in the CPE database, which references a specific version of a piece of software, as described in Figure 4. We use the CPE URIs themselves as a data source and not the CPE database they point to because they are richer and more up-to-date: as shown in Figure 5 there is an ever-increasing number of "dead" CPE URIs entries referenced in vulnerabilities metadata that do not actually exist in the CPE



CVE-2019-12370
Published on 03/18/2020
The Spark application through 2.0.2 for Android allows XSS via an event attribute and arbitrary file loading via a src attribute, if the application has the READ_EXTERNAL_STORAGE permission.
Published at disclosure by Mitre
Published after security analysis by NVD
CPE URI
cpe:2.3:a:readdle:spark:2.0.2:*:*:*:*:android:*:*
Field 'vendor' of CPE URI
readdle
Field 'product' of CPE URI
spark
Field 'target' of CPE URI
android

Figure 4: Our CPE URI-based whitelist is constructed from fields extracted from the URI included in the vulnerability metadata.

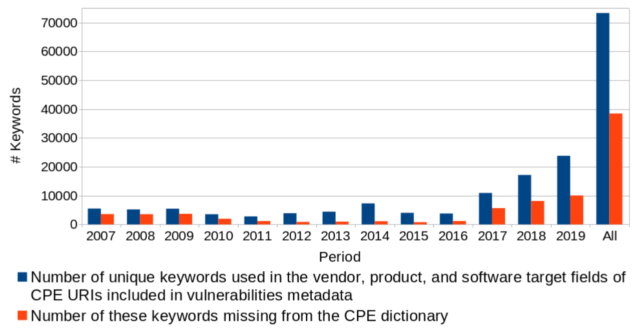


Figure 5: Historical rate of software names used in vulnerabilities CPE URIs that are missing from the CPE dictionary.

CWE Weaknesses
Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)
Stack-based Buffer Overflow
Use After Free
UNIX Symbolic Link (Symlink) Following
Extracted keywords
a after based basic buffer following free html improper in link neutralization of overflow page related script stack symbolic symlink tags unix use web xss

Table 3: Examples of CWE weaknesses and keywords extracted from them.

database. Fields extracted from the CPE URI are the software vendor, software product, and target software.

Our second whitelist is based on the CWE database. CWE is a list of common software and hardware security weaknesses, aiming to serve as a common vocabulary to describe similar vulnerabilities. We extracted all terms used in CWE titles in order to use them as a whitelist. Table 3 shows several examples of CWE weaknesses and the keywords we extracted from them.

Both CPE URI and CWE datasets are journaled, allowing us to include all entries published up until the time of disclosure of the evaluated vulnerability. We evaluate the impact of both whitelists (separately and together) in Section 4.

Word count	***Attack Vector* value**			
"local"				
	Physical	Local	Adjacent	Network
0	5	62	11	556
1	0	53	1	6
"document"				
	Physical	Local	Adjacent	Network
0	5	98	12	545
1	0	17	0	17
"compiler"				
	Physical	Local	Adjacent	Network
0	5	115	12	505
1	0	0	0	56
TOTAL				
	5	115	12	562

Table 4: Conditional distributions for various keywords and the value of CVSS field *Attack Vector* for vulnerabilities disclosed between January 1st, 2016 and April 1st, 2016.

3.3.2 Conditional entropy sorting. Our second approach to dimension reduction is not based on domain-specific knowledge but on information theory. We use *conditional entropy* to sort words by how much prediction power they provide. In this approach we consider a CVSS field as a random variable to be predicted, and the count of a word in the vulnerability description as a random variable whose value is already known. Table 4 depicts the distribution of results for the CVSS field *Attack Vector* for 694 vulnerabilities disclosed between January 1st, 2016 and April 1st, 2016, and the related conditional distributions for occurrences of words "local", "document", and "compiler" in the associated vulnerabilities descriptions. We can see that the word "compiler" has a low predictive power on *Attack Vector*, as the conditional distribution remains close to the original whether the word is present or absent. Conversely, the word "local" has more predictive power as its presence completely changes the distribution, with the most probable value switching from *Network* to *Local*. Conditional entropy, described in Equation 1, provides a synthesis of this difference by computing the entropy of one random variable when another one is known. All entropy computations in this work are done in base 2 with results expressed in bits.

$$H(Y|X) = -\sum_{x\in\mathcal{X},y\in\mathcal{Y}} p(x,y)\log\frac{p(x,y)}{p(x)} \quad (1)$$

Once we have computed the conditional entropy of a CVSS field for every word in our index, we can sort the associated dimensions: the words with the lowest conditional entropy are the more predictive dimensions. We can choose a number of dimensions we wish to retain and discard the rest.

There are two theoretical drawbacks to this approach. First the mutual information between dimensions is not taken into account. The first and second most predictive dimensions might be very predictive on their own but the second might not provide a lot

Attack Vector (AV)	
Metric Value	**Numerical Value**
Network (AV:N)	0.85
Adjacent (AV:A)	0.62
Local (AV:L)	0.55
Physical (AV:P)	0.2

Table 5: Enumerated members for CVSS V3 field *Attack Vector* and their associated numerical values as described by the CVSS 3.0 and 3.1 specifications.

more information on top of the first one. The algorithm is linear in the number of dimensions and properly handling mutual information would make it quadratic as the conditional entropy for each non-selected dimension would need to be recomputed after each dimension selection. However not handling mutual information is not a problem in practice as the dimensions are used as an input to a regression model. Dimension sorting is only used to retain a tractable number of dimensions, and as long as enough predictive dimensions have been retained, the regression model will be able to provide reasonable results.

Second, the number of dimensions we choose to retain is an hyperparameter. In practice, as this hyperparameter is the only one in the whole analysis pipeline, our hyperstate is small enough to be explored thoroughly, as we see in Section 4.

3.4 Regression Modeling on CVSS Vectors

Once we have a tractable number of dimensions to work with, we can train a model with it. We consider each CVSS vector component, or *field*, as an independent problem and we train a different model for each of them. Each CVSS vector component is valued using a multiple choice enumeration, as described by Table 5. An intuitive approach would be to use multinomial logistic regression or multinomial ordinal regression to predict these enumerated fields. However, multinomial regression models are not straightforward to train, with iterative analytical methods such as Iteratively Reweighted Least Squares (IRLS) [21] or iterative gradient-based solvers such as L-BFGS-B [26]. All iterative methods inherently require one or two new hyperparameters to be added to the hyperstate (either the acceptable error threshold before stopping or the maximum number of steps, or both), and a lot of solvers require additional hyperparameters specific to their approach.

The CVSS specification gives us an interesting way of simplifying our modeling. All CVSS V2 fields values and all but one CVSS V3 fields have a numerical value associated with each enumerated value. This numerical value is used in the calculation of the CVSS severity score. Table 5 shows the numerical value for each enumerated values of the CVSS field *Attack Vector*.

Instead of using multinomial regression to predict the value of a CVSS field enumeration, we can use linear regression to predict the numerical value for the CVSS field. From the predicted numerical value, we then select the enumerated value whose associated numerical value is the closest to the prediction. A linear model can be trained analytically using Ordinary Least Squares (OLS) without requiring any new hyperparameter.

One CVSS V3 field, *Scope*, is a binary field (with possible values *Changed* or *Unchanged*) with no associated numerical value. For this field we associate *Unchanged* and *Changed* to -1 and 1 respectively then treat it in the same way as every other field. This reduces to a binary logistic regression problem followed by selecting the most probable outcome of the two, without considering the odds.

4 EVALUATION

4.1 Experimental Setup

We analyzed all 33807 CVE vulnerabilities disclosed between January 1st, 2018 and January 1st, 2020, using all 70172 CVE vulnerabilities disclosed between January 1st, 2007 and December 31st, 2017 as past historical data. This experimental setup simulates the behavior of a production system put online on January 1st, 2018, initially fed with historical information from eleven years before, which then monitors all newly disclosed vulnerabilities continuously for the next two years. Each vulnerability is analyzed using the data available at its disclosure day only, as described in Figure 3. Using the data shown in Figure 2, we decided to use a fixed *metadata publication delay* for all vulnerabilities which we set at 60 days. The choice of 60 days ensures analysis conditions that are overall realistic (albeit simplified) but strictly worse than any recorded median case, and close to the worst recorded 9$^{\text{th}}$ decile. Therefore if our analysis technique performs well during evaluation, we can be highly confident that it will perform as well or better in the real world.

Seven configurations of our analysis pipeline were evaluated: three are whitelist-based (CPE, CWE, and both combined together), four are based on conditional entropy (number of retained dimensions was set to 100, 500, 1000, and 5000). All these configurations are evaluated for CVSS V2 and V3 prediction. In order to optimize our computing resources usage, our experiments are simulated assuming a weekly retraining of the regression model (simulating a daily retraining would require 7 times more resources). Nevertheless in Section 4.6 we evaluate the impact of going from weekly to daily retraining for the most promising configuration. All the code and data used for our experiments are available at [16].

4.2 Prediction for Individual Fields

Figures 6 and 7 show the raw success rate for each field and predictor: that is, the number of correct predictions over the number of total predictions. From this data we can make a few conclusions.

- Dimension reduction using a whitelist based on CPE URIs exhibits overall worse accuracy than all other approaches, even when combining it with the CWE whitelist.
- Dimension reduction using conditional entropy with a number of retained dimensions between 500 and 1000 exhibits better accuracy than all other approaches.
- Some fields are more difficult to predict than others. In particular, the Confidentiality, Integrity and Availability fields have lower accuracy than other fields, especially in CVSS V3.

Success rates show how the predictors are working "out of the box", but not the best possible predictor that can be constructed through the prediction technique, as it does not take into account

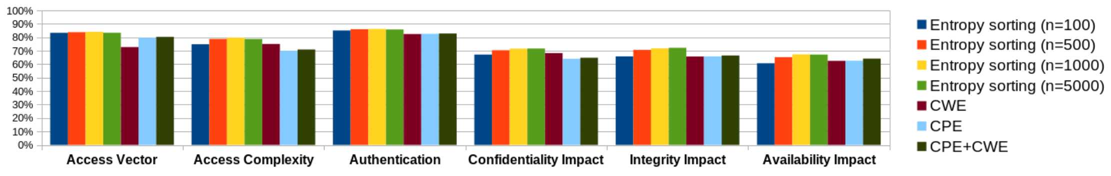


Figure 6: Success rate for individual CVSS V2 field predictions.

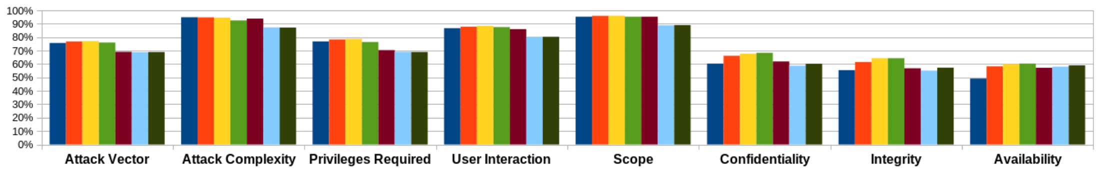


Figure 7: Success rate for individual CVSS V3 field predictions.

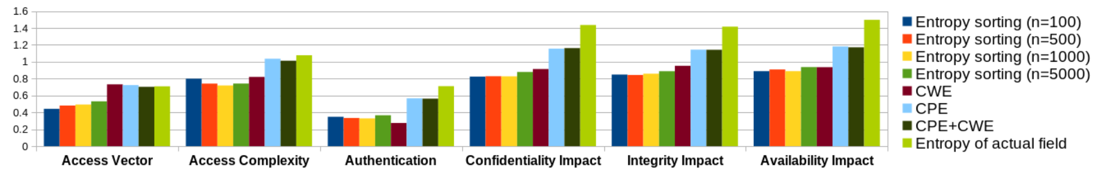


Figure 8: Conditional entropy for individual CVSS V2 field predictions (lower is better).

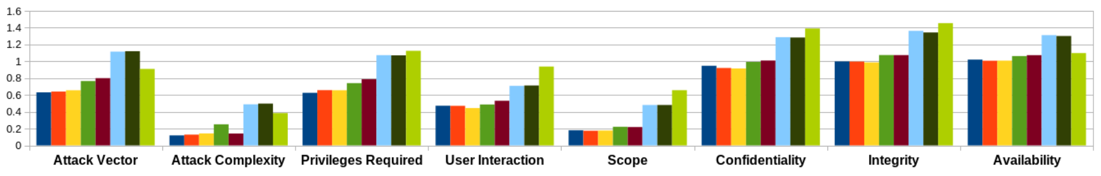


Figure 9: Conditional entropy for individual CVSS V3 field predictions (lower is better).

"permuted" predictions. For example, it is possible to use a predictor guessing wrong 100 % of the time to construct another predictor that guesses right 100 % of the time.

To thus further evaluate our predictors we can use conditional entropy again (this time as an evaluation tool) in order to measure the predictive power of each configuration on individual CVSS fields. By computing the original entropy of a given CVSS field, and then computing the conditional entropy of this field when knowing a predicted value of it, we can measure how much information is gained through the prediction. Figures 8 and 9 show the conditional entropy for each CVSS field and each prediction pipeline configuration, for CVSS V2 and CVSS V3 respectively. Each column group describes one CVSS field, with each column measuring the entropy of this field conditioned to the predicted value for this field according to the given configuration (the lower entropy the better). The last column of each group shows the entropy of the actual CVSS field, to serve as a baseline for comparison. We can see that on some fields the conditional entropy is even higher than the initial entropy: this can be interpreted as performing worse than a random guess (weighted using the past frequencies of the different members of the CVSS field). However our best configurations all have significantly lower conditional entropy than the real entropy of the CVSS field, which indicates that meaningful information was gained through the predictor.

Configuration	50 %		80 %		99 %	
Entropy sorting (n=100)	0	2.4	-1.5	3	-5	5.4
Entropy sorting (n=500)	0	1.8	-1.5	2.6	-5.4	5.4
Entropy sorting (n=1000)	0	1.5	-1.5	2.5	-5.8	5.4
Entropy sorting (n=5000)	0	1.5	-1.7	2.5	-6.4	5.4
CWE	-0.3	1.7	-1.9	2.8	-5.4	5.4
CPE	-0.6	1.7	-2.5	2.6	-7.8	5.4
CPE + CWE	-0.7	1.5	-2.6	2.5	-8.5	5.4

Table 6: Error intervals for CVSS V2 score prediction for 50 %, 80 % and 99 % of the vulnerabilities in the evaluation dataset.

4.3 Prediction for the CVSS Severity Score

After making a prediction for each CVSS field for a given vulnerability, we can assemble a complete, predicted CVSS vector and

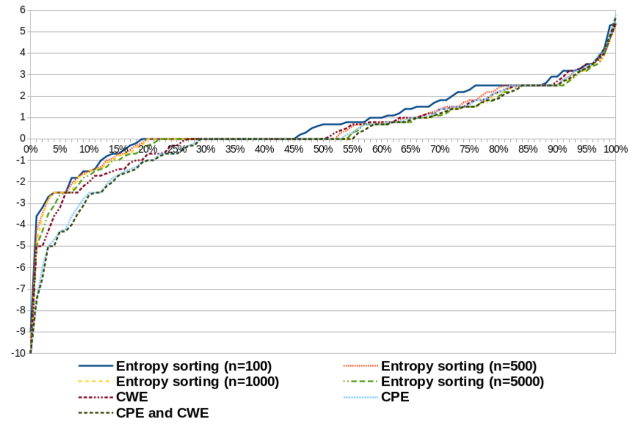


Figure 10: Error distribution for CVSS V2 score prediction.

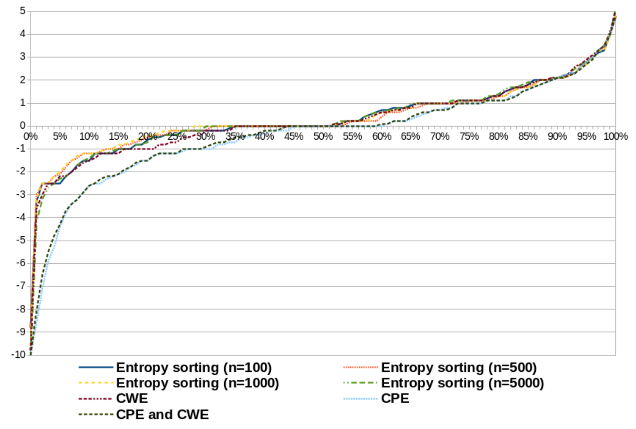


Figure 11: Error distribution for CVSS V3 score prediction.

Configuration	50 %		80 %		99 %	
Entropy sorting (n=100)	-0.4	1.1	-1.4	2.1	-3.5	4.5
Entropy sorting (n=500)	-0.2	1.1	-1.2	2.1	-3.5	4.5
Entropy sorting (n=1000)	-0.2	1.1	-1.2	2.1	-3.7	4.5
Entropy sorting (n=5000)	-0.4	1.1	-1.4	2.1	-5.4	4.6
CWE	-0.7	1.1	-1.5	2.1	-5.4	4.7
CPE	-1.2	1	-2.6	2.1	-9.8	4.5
CPE + CWE	-1.1	1	-2.6	2.1	-9.8	4.5

Table 7: Error intervals for CVSS V3 score prediction for 50 %, 80 % and 99 % of the vulnerabilities in the evaluation dataset.

compute its severity score using the standard CVSS computation rules (V2 or V3). We can then compare this predicted severity to the actual severity of the vulnerability, computed from the actual CVSS vector for the vulnerability. We define the *severity prediction error* as $PredictedSeverity - ActualSeverity$, giving us a value between -10.0 and $+10.0$. A severity prediction error of 0 is a perfect prediction. Any value above 0 is a *false positive*, with the vulnerability being *less* severe than predicted. Any value below 0 is a *false negative*, with the vulnerability being *more* severe than predicted. There is an inherent trade-off between false positives and false negatives, and different organizations can have different preferences about this trade-off: alert systems exhibiting false negatives can leave an organization vulnerable to undetected threats, while too many false positives can lead an alert system to become unusable in practice.

Figures 10 and 11 show the distribution of the severity error prediction in our evaluation dataset, while Tables 6 and 7 show the error intervals for 50 %, 80 %, and 99 % of our evaluation dataset respectively. From these results we can make the following conclusions:

- Dimension reduction based on conditional entropy sorting provides systematically better accuracy than dimension reduction based on whitelists. In particular all whitelists based on CPE URIs provides strictly less accuracy than all other approaches. Our whitelist based on the CWE framework provides better results, closer in accuracy to entropy sorting but still outperformed by the best entropy sorting configurations. Interestingly, combining CWE and CPE whitelists into one decreases accuracy compared to either whitelists, making it the worse-performing configuration.
- Regarding conditional entropy sorting, the number of dimensions kept after sorting (our sole hyperparameter) does have an impact on accuracy. In both CVSS V2 and V3, keeping the number of dimensions between 500 and 1000 is important in order to get the best prediction results. When predicting the CVSS V3 severity score, accuracy differences between 100, 500 and 1000 retained dimensions are nearly indistinguishable. When predicting CVSS V2 severity scores, retaining 100 dimensions provides more false positives than 500 or 1000. However it is possible this is an artifact of our evaluation dataset (this could be checked in the future by reproducing our experiments using vulnerabilities disclosed from 2020 and later years), especially given that there are no such differences when predicting CVSS V3.
- In nearly all cases the prediction error for CVSS V3 is lower than for CVSS V2. This was surprising to us as CVSS V3 includes eight fields while CVSS V2 only includes six: this alone should make it more likely to make more errors when predicting a full CVSS V3 vector. Our hypothesis is that CVSS V3 computation rules are more likely to give closer severity scores to similar but not identical vectors.

4.4 Results Explicability

To maintain explicability, our prediction pipeline is able to show which weighted keywords were used to make a prediction. Table 8 shows an example of how explicability can be maintained throughout the prediction pipeline. CVE-2018-17625 is a vulnerability affecting Foxit Reader disclosed on 01/23/2019. For two CVSS V3 fields (*Attack Vector* and *User Interaction*) we show the top four keywords used to predict their value. We can see that the presence of the word "remote" was the biggest factor when making the decision to predict the value *Network* for *Attack Vector* (meaning vulnerability exploitation can be accomplished remotely). Conversely, the word "file" was an important factor to predict the value *Required* for *User Interaction* (meaning vulnerability exploitation requires a user to do a specific action).

Description				
This vulnerability allows remote attackers to execute arbitrary code on vulnerable installations of Foxit Reader 9.1.0.5096. User interaction is required to exploit this vulnerability in that the target must visit a malicious page or open a malicious file. The specific flaw exists within the handling of the setInterval() method. The issue results from the lack of validating the existence of an object prior to performing operations on the object. An attacker can leverage this vulnerability to execute code in the context of the current process. Was ZDI-CAN-6438.				
Attack Vector	**Actual:** Network, **Predicted:** Network			
Keyword	remote	required	file	interaction
Weight	0.059	0.037	-0.029	0.025
User Interaction	**Actual:** Required, **Predicted:** Required			
Keyword	file	page	malicious	interaction
Weight	-0.287	-0.021	-0.020	-0.019

Table 8: Top four keywords used to predict CVSS V3 fields *Attack Vector* and *User Interaction* for vulnerability CVE-2018-17625 disclosed on 01/23/2019, using a CWE whitelist for dimension reduction.

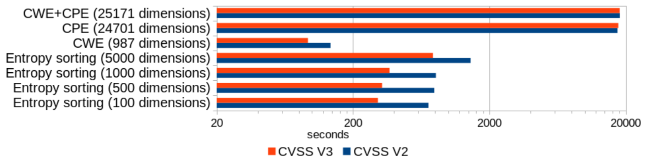


Figure 12: Duration in seconds of the complete analysis pipeline at date 2019-01-01. Time scale is logarithmic.

4.5 Performance Impact of Dimension Reduction

The performance of the pipeline is not a goal in itself for us, however we want to ensure that our architecture is suited for daily retraining. That means a complete retraining of all our models followed by an inference for all new vulnerabilities disclosed on a new day should not take more than a fraction of a day. Figure 12 shows the duration of the pipeline (in seconds, using a logarithmic scale) for each configuration, at date 2019-01-01, assuming a complete retraining followed by an analysis of all vulnerabilities disclosed on that day. All computations were done on Dell PowerEdge C6420 servers with Intel Xeon Gold 6130 CPUs (Skylake, 2.10 GHz, 2 CPUs/node, 16 cores/CPU) and 192 GiB of RAM. All performance experiments were run three times: duration was very stable with half of variations below 1 % and all of them below 3 %. The quickest configuration was based on the CWE whitelist, with an average training duration of 1 min 33 s for CVSS V3 and 2 min 16 s for CVSS V2, while the longest configuration was based on the CPE whitelist, with a duration close to five hours. Meanwhile, entropy sorting configurations ran during 5 to 25 minutes depending on the number of retained dimensions. All configurations are therefore suited for daily retraining if deemed necessary. A production implementation of the pipeline could likely be faster than our current prototype.

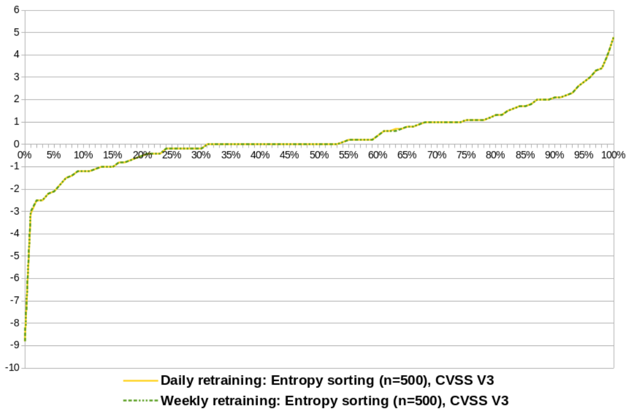


Figure 13: Impact of daily and weekly retraining on the error distribution for CVSS V3 score prediction.

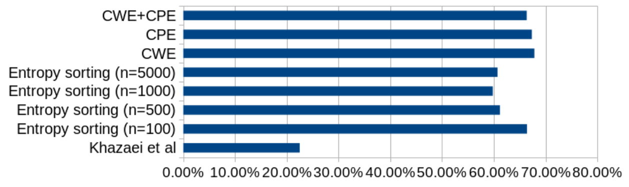


Figure 14: Error rate (as defined by Khazaei et al) for CVSS V2 score prediction.

4.6 Daily Retraining vs Weekly Retraining

As described in Section 4.1, all experiments were run assuming a weekly retraining of the prediction model. This was an experimental constraint as we did not have the computing resources to simulate all experiments assuming a daily retraining (which consumes seven times more resources). However performance measurements from Section 4.5 suggest that daily retraining of a production system is feasible. Therefore we ran the most promising configuration twice (dimension reduction through entropy sorting with 500 retained dimensions) assuming both weekly and daily retraining of the model for CVSS V3 prediction. This impacted the score of 1371 vulnerabilities, around 4 % of our evaluation dataset. As shown in Figure 13 this has negligible impact on the predicted severity error distribution using our experimental setup. A similar experiment done with CVSS V2 on a subset of the same dataset gave similar results. One should note that our choice to simulate a metadata publication delay of 60 days for all vulnerabilities could hide some of the impact of the training frequency compared to a production system relearning a new model as soon as new information becomes available. However in all cases accuracy of such production systems should at least be identical or even better than our results, for the reasons described in Section 3.1.

5 DISCUSSION AND LIMITATIONS

Our prediction pipeline is designed with predictability and explicability in mind and this has an impact on accuracy. Figure 14 compares our approach with the more elaborated machine learning techniques used by Khazaei et al. The comparison has limits as they used the now unavailable OSVDB dataset on years 2012 and 2013 (using training data starting from 2004) while we used the NVD dataset on years 2018 and 2019 (using training data starting from 2007). Moreover the error rate metric they proposed does not allow differentiation between small and large errors, a problem we detailed in Section 2. Nevertheless, their techniques are indisputably more effective at predicting CVSS scores than ours. This raises a number of questions for an organization willing to deploy a CVSS prediction pipeline: do CVSS vectors have inherent value or are they just a mean to compute a severity score? How to balance worst-case accuracy and average accuracy? Is decision explicability important? Different organizations will not have the same answers to these questions, leading them to different architectures. Maintaining two prediction pipelines, one explicable and one more accurate, would allow raising alerts when their decisions differ. In the future we hope to improve the accuracy of our prediction pipeline to get closer to Khazaei et al, while still preserving the predictability and explicability properties making our current architecture novel.

6 CONCLUSION

We introduced a method to automatically predict CVSS vectors for newly disclosed vulnerabilities, relying only on their human-readable description. Our results are promising, as a simple technique brings results that are accurate enough to be useful while providing decision explicability, a missing property in the state of the art. As discussed in Section 1, our CVSS prediction technique is a first step toward automated reaction to new vulnerability disclosures. In future work we intend to use this prediction pipeline to build a complete, automated threat analysis system at disclosure. CVSS is a measure of severity and not risk: a threat analysis system also needs to assess how much risk does a vulnerability pose to a specific information system. The automated risk analysis and reaction mechanisms made possible by our technique could allow faster and cheaper reaction to n-day vulnerabilities, and thus become invaluable tools for security engineers defending information systems against day-to-day threats.

ACKNOWLEDGMENTS

Clément Elbaz's PhD Grant is co-funded by the Brittany Council. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

REFERENCES

[1] 2020-08-04. A Complete Guide to the Common Vulnerability Scoring System: Version 2.0. https://www.first.org/cvss/v2/guide.
[2] 2020-08-04. AWS WAF - Web Application Firewall. https://aws.amazon.com/waf/.
[3] 2020-08-04. Cloudflare - Inside Shellshock: How hackers are using it to exploit systems. https://blog.cloudflare.com/inside-shellshock/.
[4] 2020-08-04. Cloudflare - Stopping SharePoint's CVE-2019-0604. https://blog.cloudflare.com/stopping-cve-2019-0604/.
[5] 2020-08-04. Cloudflare Web Application Firewall. https://www.cloudflare.com/waf/.
[6] 2020-08-04. Common Vulnerabilities and Exposures (CVE). https://cve.mitre.org/.
[7] 2020-08-04. Common Vulnerability Scoring System. https://www.first.org/cvss/.
[8] 2020-08-04. Common Vulnerability Scoring System v3.0: Specification Document. https://www.first.org/cvss/v3.0/specification-document.
[9] 2020-08-04. Common Vulnerability Scoring System v3.1: Specification Document. https://www.first.org/cvss/v3.1/specification-document.
[10] 2020-08-04. CVE and NVD Relationship. https://cve.mitre.org/about/cve_and_nvd_relationship.html.
[11] 2020-08-04. CWE - Common Weakness Enumeration. https://cwe.mitre.org/.
[12] 2020-08-04. Google Cloud Armor. https://cloud.google.com/armor/.
[13] 2020-08-04. National Vulnerability Database. https://nvd.nist.gov/.
[14] 2020-08-04. NVD - CPE. https://nvd.nist.gov/products/cpe.
[15] 2020-08-04. Security Content Automation Protocol. https://csrc.nist.gov/projects/security-content-automation-protocol.
[16] 2020-23-06. Firres. https://gitlab.inria.fr/celbaz/firres_ares.
[17] Leyla Bilge and Tudor Dumitraş. 2012. Before We Knew It: An Empirical Study of Zero-day Attacks in the Real World. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS' 12)*.
[18] Sandy Clark, Stefan Frei, Matt Blaze, and Jonathan Smith. 2010. Familiarity Breeds Contempt: The Honeymoon Effect and the Role of Legacy Code in Zero-day Vulnerabilities. In *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC '10)*.
[19] Stefan Frei, Martin May, Ulrich Fiedler, and Bernhard Plattner. 2006. Large-scale Vulnerability Analysis. In *Proceedings of the 2006 SIGCOMM Workshop on Large-scale Attack Defense (LSAD '06)*.
[20] Leonid Glanz, Sebastian Schmidt, Sebastian Wollny, and Ben Hermann. 2015. A Vulnerability's Lifetime: Enhancing Version Information in CVE Databases. In *Proceedings of the 15th International Conference on Knowledge Technologies and Data-driven Business (i-KNOW '15)*.
[21] Paul W. Holland and Roy E. Welsch. 1977. Robust regression using iteratively reweighted least-squares. *Communications in Statistics - Theory and Methods* 6, 9 (1977).
[22] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*.
[23] Jay Jacobs, Sasha Romanosky, Benjamin Edwards, Michael Roytman, and Idris Adjerid. 2019. Exploit Prediction Scoring System (EPSS). In *Black Hat 2019*. http://i.blackhat.com/USA-19/Thursday/us-19-Roytman-Predictive-Vulnerability-Scoring-System-wp.pdf
[24] Karen Spärck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation* 28 (1972).
[25] Atefeh Khazaei, Mohammad Ghasemzadeh, and Vali Derhami. 2016. An automatic method for CVSS score prediction using vulnerabilities description. *Journal of Intelligent & Fuzzy Systems* 30, 1 (2016).
[26] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. 1997. Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization. *ACM Trans. Math. Softw.* 23, 4 (1997).