Automated Risk Analysis of a Vulnerability Disclosure Using Active Learning

Clément Elbaz, Louis Rilling, Christine Morin

Automated Risk Analysis of a Vulnerability Disclosure Using Active Learning

Clément Elbaz[1,2], Louis Rilling[2] and Christine Morin[3]

[1]Univ Rennes, Inria, CNRS, IRISA, France
[2]DGA, France
[3]Inria, France

Abstract

Exhaustively listing the software and hardware components of an information system is non-trivial. This makes it even harder to analyze the risk created by a vulnerability disclosure in the context of a specific information system. Instead of basing the risk analysis of a newly disclosed vulnerability on a possibly obsolete list of components, we focus on the security team members tasked with protecting the information system, by studying how Chief Information Security Officers (CISOs) and their subordinates actually react to vulnerability disclosures. We propose to use active learning to extract the conscious and unconscious knowledge of an information system's security team in order to automate the risk analysis of a newly disclosed vulnerability for a specific information system to be defended.

Keywords

Security, CVE, Machine Learning, Active Learning

1. Introduction

Disclosure is the most critical part of a vulnerability life cycle. Bilge *et al.* [1] showed a five orders of magnitude increase in the usage of vulnerabilities before and after their disclosure. In the midst of this increasing risk, 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 many systems vulnerable in practice.

We propose an automated system evaluating, for a newly disclosed vulnerability, the necessity to alert an on-call security operator regarding a new risk in the context of a specific information system to be defended. To this end we intend to let an expert (a CISO or her subordinates) participate in the training of the prediction system by annotating past vulnerabilities to indicate whether she would have wished being alerted about a similar recent disclosure. Unlike a SIEM [2], our system does not need to be connected to the monitored information system, as we focus exclusively on public events (vulnerability disclosures) and how they match our expert's interests. We should however fulfill the following constraints:

C&ESAR Conference, November 16–17, 2021, Rennes, France
clement.el-baz@def.gouv.fr (C. Elbaz); louis.rilling@irisa.fr (L. Rilling); christine.morin@inria.fr (C. Morin)

1. We want to make the best use of past historical vulnerability data in order to predict how to react to future vulnerabilities.
2. We want to make the best use of the limited time of security experts by allocating a fixed *annotation budget* for the training and *selecting the most relevant vulnerabilities to be annotated*.
3. In order to keep the operator's trust in the prediction system we want both the vulnerability selection process during the training stage, and the vulnerability analysis process during the inference stage, to be explicable.

In Section 2 we discuss background and related work. In Section 3 we present our approach. In Section 4 we propose a preliminary evaluation of our technique. We discuss our results in Section 5 and conclude in Section 6.

2. Background and Related Work

The process of disclosing new vulnerabilities is coordinated by the *Common Vulnerabilities and Exposures* (CVE) system overseen by Mitre's Corporation [3]. 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* [4] or SCAP data feeds [5]. Only then they will be annotated by multiple security experts. As of September 2021, more than 160 000 vulnerabilities have been disclosed by CVE and analyzed by NVD. This has allowed the research community to explore the vulnerability life cycle at scale [6, 7, 8, 9] and converge on three insights. First, the relationship between attacks and vulnerabilities follows a power-law, with a minority of vulnerabilities being exploited in a majority of attacks, and most vulnerabilities never being exploited at all [10, 11]. The actual rate of vulnerability exploitation has been subject to discussion, with figures going from 15 % [10] to 1.3 % [12] having been reported in the literature. Second, too many vulnerabilities are disclosed at any time for them to be treated with an equal level of urgency. More than 18000 vulnerabilities were disclosed by CVE [3] in 2018 and 2019, around 50 vulnerabilities per day on average. Prioritization is required. Third, as mentioned in Section 1, usage of a vulnerability in the wild can increase as high as *five orders of magnitude* between before and after vulnerability disclosure [1].

These three facts taken together led the community to a conclusion: *vulnerability exploitation in the wild*, both in the present and the future, is the most important property of a vulnerability to predict. By properly predicting exploitation in the wild, one can efficiently prioritize limited mitigation resources while still preventing most attacks. Starting from 2010, most efforts in the community were devoted to finding accurate predictors of exploitation, with attempts focusing on CVSS [13, 14, 15, 16], OSVDB [17, 18], exploit black-markets [16], social networks [12, 19] and online forums [20]. These prediction efforts culminated with the release in 2019 of the *Exploit Predicting Scoring System* (EPSS) [21], which uses many data sources to train a regression model answering a simple question: what is the probability of a vulnerability being exploited in the wild in the twelve months following its disclosure?

All of these models are based on data usually generated after disclosure, such as human discussion or exploit trading happening after the vulnerability has already gone public for some time. While some of these models can provide reasonable results days after the disclosure,

the ability to predict the danger posed by a vulnerability in the minutes or seconds after its disclosure has not improved significantly. This is a problem as some major vulnerabilities such as *Shellshock* [22] were exploited within one hour of their disclosure. Therefore our own previous work [23] has focused on predicting multiple properties of a vulnerability, such as the affected software [24] or its CVSS vector [25] immediately after disclosure. Moreover, for a specific organization, exploitation in the wild is not a sufficient criterion to act, as many vulnerabilities affect software that is not used in the organization's information systems.

Cataloging an information system exhaustively is hard. Software has many forms and many locations: while software binaries reside on hard disks, firmware is embedded inside hardware, and uncompiled source code can be interpreted at run time. Software can be downloaded on the fly over the network without touching the disk, such as when browsing the web with Javascript enabled. Hardware can be vulnerable [26][27] and is not easily updated or replaced. Databases such as the *Common Platform Enumeration* (CPE) [28] (which references all software and hardware ever affected by a vulnerability) can in theory be used to track and monitor all these components but in practice discrepancies between the real world and CPE entries (which are authored manually) quickly accumulate to the point of becoming unmanageable [29]. This problem also impacts software versioning: as an example the Debian security team [30] is known to backport security fixes into Debian packages without waiting for official releases of the upstream software, creating a Debian-specific version number while doing so. While this practice should be lauded for improving the timeliness of Debian security updates, it adds confusion about how a given version of a software is supposed to behave.

The evolution of cybersecurity threats led organizations to pursue continuous protection through the creation of *Security Operation Centers* (SOCs) dedicated to ensuring their ongoing safety. Ideally, every organization would benefit from being protected by a SOC, either internal or outsourced. However, this is a challenging goal in practice, as hiring security analysts is difficult and expensive [31, 32] and the profession has a documented history of elevated stress and burnout [33, 34, 35, 36, 37]. Therefore, at a global level it is neither possible to hire more security analysts nor to give them much more work to do. These facts taken together mean that the human component of cybersecurity defense is globally saturated, and the only way to meet the ever-increasing demand for real-time cybersecurity defense is through automation.

Active learning [38] is a hybrid class of machine learning where training is an interactive process between an automated learner and an oracle such as a human expert. The learner starts with a set of unlabeled data. It then has to solve two learning problems: the first is to select the next sample to be submitted to the oracle for labeling, and then to use both the labeled and unlabeled data to make predictions. An important characteristic of active learning is training interactivity: each label provided by the oracle should impact which sample should be submitted next. Active learning is an interesting choice for information security as the same channel between the learner and the oracle can be used for both training and evaluation: once the system is trained, random unlabeled samples can be sent to both the learner and the oracle to compare their answers. The oracle, the learner, the data and its annotations can all remain in one security context, avoiding the need to share sensitive datasets between multiple security contexts. An example of active learning for security is *ILAB* [39], a framework designed by Beaugnon *et al* aiming to facilitate the annotation of intrusion detection datasets.

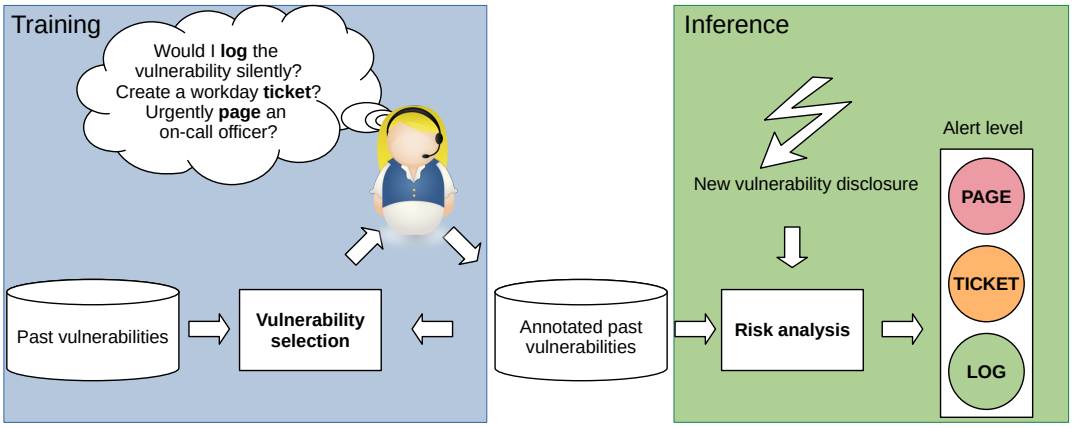

Figure 1: Proposed active learning architecture to estimate the risk-level evolution.

3. Proposed Approach

We propose an active learning architecture based on the work by Settles *et al* [42], which we selected because of its simplicity and explicability properties. Our architecture is depicted in Figure 1. It is composed of a *training phase* during which an expert and the system train together and an *inference phase* during which the system monitors new vulnerabilities and evaluates the risk that they create. Since the active learning architecture originally proposed by Settles *et al* is deterministic and only handles static datasets, we made slight modifications to the scheme (detailed in Sections 3.6 and 3.7) to handle the dynamic nature of an ongoing flow of vulnerability disclosures and support a controllable amount of randomness during the training process.

During the training phase the system iteratively selects a relevant past vulnerability, submits it to the expert through the user interface, who then annotates it by choosing an appropriate alert level for the vulnerability. The training phase is actually split in two parts. First, the *offline training phase* happens during the deployment of a production prediction system at a fixed point in time, and lets the prediction system and the expert create together an initial knowledge base by letting the prediction system select any past vulnerability at that point in time, with the system having full control on the vulnerability selection process. Second, the *online training phase* is a continuous interaction between the prediction system and the expert over time, discretized into *periods* such as weeks. Every time a new period starts, the prediction system iteratively selects new vulnerabilities among the ones disclosed during the last period and makes the expert annotate them. Each of these phases has a separate annotation budget, with the offline phase requiring a one-off effort from the expert and the online phase requiring a continuous effort over time. These two phases of the training are important, as they allow the prediction system to acquire both long-term context on past vulnerability data as well as

awareness of the latest vulnerability disclosures.

Once the expert considers that the system has been made reliable enough through offline and online training (we come back to this in Section 4), the inference phase can begin. The system now monitors new vulnerability disclosures in near real time and chooses an appropriate alert level for them using the knowledge base it has constructed together with the expert.

The processing of a vulnerability is made of several building blocks, some of them common to both the training and inference phases, and some specific to one or the other, as detailed below. During the inference phase, evaluating the risk created by a new vulnerability is done by comparing it to similar, annotated vulnerabilities. To this end a *similarity metric* between vulnerabilities is needed. This similarity metric is made possible by viewing vulnerabilities as euclidean vectors through a *featurization stage*.

During the training phase, selecting relevant vulnerabilities to be annotated by the expert is based on the same building blocks. Our similarity metric is used to compute a *similarity matrix* for the entire past vulnerabilities dataset, which then lets the learning system compute an *information density metric* for every past vulnerability. Selecting a vulnerability to be annotated by the expert is done by combining this information density metric with an *uncertainty metric* computed by simulating an alert decision process for every vulnerability while measuring the confidence of the prediction.

In the rest of this section we first present the concepts necessary for both training and inference: in Section 3.1 we describe the possible alert levels for a vulnerability and the user interface used by the expert. In Section 3.2 we discuss the featurization stage of our architecture. In Section 3.3 we discuss our choice of similarity metric. The actual alert decision process used during the inference phase is then presented in Section 3.4. We finally cover the remaining concepts necessary for the training phase. In Section 3.5 we present our choice of uncertainty metric. We then discuss concepts proposed by Settles *et al* [42] that we use: first the similarity matrix and information density metric in Section 3.6, then in Section 3.7 the process used by the learning system to select the next vulnerability to be annotated by the expert during the training phase.

3.1. User Interface and Alert Levels

An alert level scale is needed to let both the system and the expert translate their risk analysis of a vulnerability into an actionable decision. The recipient of this alert is a human, so it makes sense for the alert scale to be human-centered. We use the scale proposed by Google in their *Site Reliability Engineering* doctrine [43]:

- **LOG:** The disclosure is *logged* in an activity log that can be consulted afterward, but no alert is actively raised to a human security operator.
- **TICKET:** The disclosure causes the creation of a *ticket* in an issue tracking system. A ticket is expected to be solved in a non-urgent manner during working hours.
- **PAGE:** The disclosure causes an on-call security operator to be *paged* immediately, even outside regular working hours. A page is expected to be treated immediately.

As shown in Figure 2, in the training phase the system selects a vulnerability to be labeled by the human expert then shows its description on screen. The expert then selects the alert level

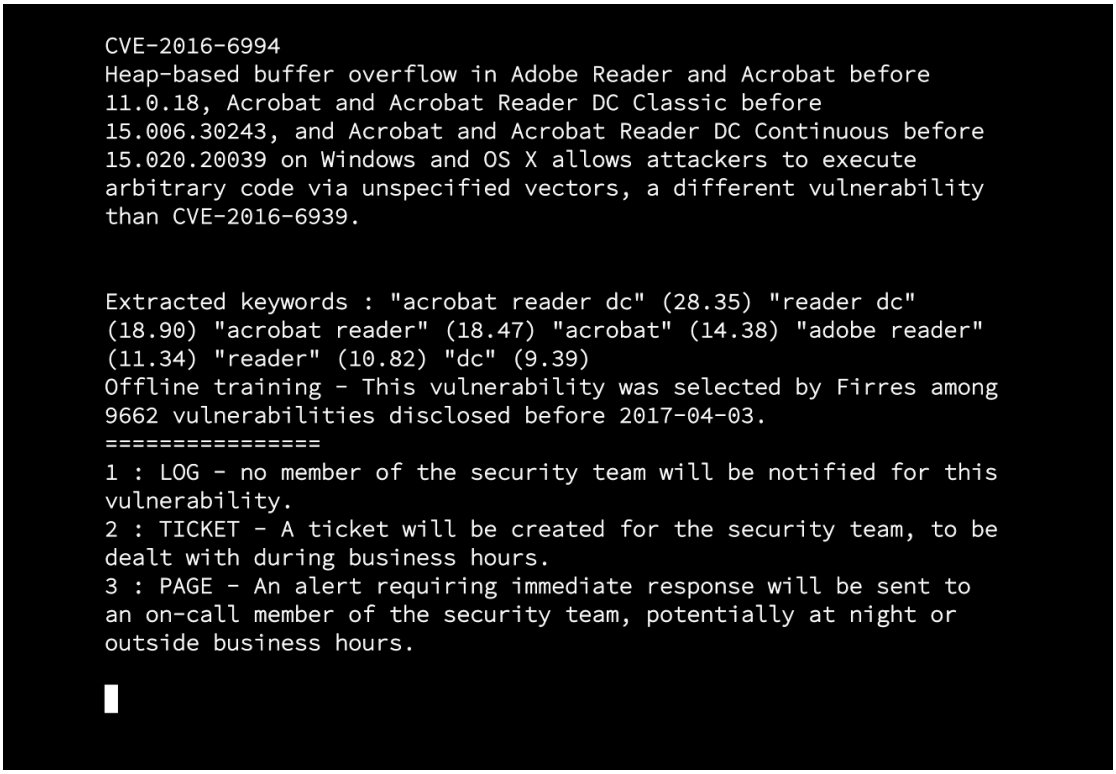

Figure 2: A screen capture of the user interface (UI) of our prototype Firres (FIRst RESponder) during the active learning phase. For a vulnerability to be labeled, the UI displays its CVE ID, its description, the weighted keyword list extracted from the description. The expert can then select an appropriate alert level by entering 1 (for LOG), 2 (for TICKET) or 3 (for PAGE).

she deems the most appropriate for the vulnerability, in the context of the system she is tasked to protect. In the inference phase, the system automatically selects the appropriate alert level for new vulnerabilities using the same alert scale.

3.2. Vulnerability as a Euclidean Vector

In order to transform the raw description of a vulnerability into a euclidean vector, we reuse the same keyword extraction pipeline as described in our past work [24]: we retain only words present in past CPE URIs then weight them using an algorithm called *TF-IDF* [40].

TF-IDF is a numerical statistic reflecting the importance of a word in a document, in the context of a corpus. It is defined in Equation 1 where t is a word and d is a document belonging to a corpus of documents D.

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D) \tag{1}$$

TF is based on the number of occurrences of the word in the document. Several formulas have been proposed, such as using the raw count directly or treating the presence or absence of

a word as a boolean event. Applying a logarithm on top of the raw count of occurrences has been argued [41] to better reflect the diminishing returns of repeating the same term several times.

IDF is defined in Equation 2:

$$\text{IDF}(t, D) = \log \frac{|D|}{|d \in D \ : \ t \in d|} \tag{2}$$

where $|D|$ is the number of documents in the corpus, and $|d \in D \ : \ t \in d|$ is the number of documents of the corpus containing the word t. TF-IDF therefore allows more specific words to have a bigger impact on the mapping than common words. This weighting is then improved using domain-specific heuristics presented in [24]. We finally use a euclidean truncation scheme to reduce the number of non-zero components for every vector. This pipeline lets us identify the affected software for a vulnerability in the form of a *sparse* euclidean vector (with a low number of non-zero vector components). Vulnerabilities affecting the same software share many non-zero components, making them similar in the vector space.

3.3. Measuring Vulnerability Vectors Similarity

As a similarity metric between vulnerability euclidean vectors we choose the *cosine similarity* that can be computed using the formula shown in Equation 3:

$$\text{cosine similarity}(\mathbf{x}, \mathbf{x'}) = \frac{\sum_{i=1}^{n} \mathbf{x}_i \mathbf{x'}_i}{\|\mathbf{x}\| \|\mathbf{x'}\|} \tag{3}$$

Cosine similarity returns a similarity between 0 and 1 for two vectors. Its main avantage is to return a similarity of zero for vectors having no common non-zero components. This is helpful to avoid creating artificial similarity between unrelated vulnerabilities. In particular, distance-based similarity metrics tend to overestimate the similarity between vectors close to origin. In our case, a vector close to origin indicates a vulnerability for which we have not extracted any meaningful keyword and therefore have no strong understanding of. Therefore having two vectors being close to origin does not necessarily means that the two related vulnerabilities are actually similar to each other. Another advantage of cosine similarity is to not require any hyperparameter, making it more explicable and stable than similarity metrics that do.

3.4. Alerting Decision Score

For every new vulnerability disclosure, the prediction system has to make a risk estimate by choosing an appropriate alert level for the vulnerability, among LOG, TICKET and PAGE. In order to do that, we had to make several hypotheses. These hypotheses can be challenged, as we discuss in Section 5.

Our first hypothesis is that the TICKET alert level is a reasonable default reaction for a vulnerability about which nothing is known. This is based on the assumption that security operators are *risk averse* and want to eventually review any vulnerability for which the prediction system could not make a meaningful decision. In the light of this hypothesis, we propose an

alert decision score in range $[-1; +1]$. It can be converted into an *alert decision* for a vulnerability v using Equation 4.

$$\text{decision}(v) = \begin{cases} LOG, & \text{if score}(v) \in [-1; -\frac{1}{3}[\\ TICKET, & \text{if score}(v) \in [-\frac{1}{3}; +\frac{1}{3}[\\ PAGE, & \text{if score}(v) \in [+\frac{1}{3}; +1] \end{cases} \tag{4}$$

Conversely, a default alert decision score is set for all annotated vulnerabilities. The alert decision score of an annotated vulnerability av is set following Equation 5.

$$\text{score}(av) = \begin{cases} -1, & \text{if the vulnerability is annotated as LOG} \\ 0, & \text{if the vulnerability is annotated as TICKET} \\ +1, & \text{if the vulnerability is annotated as PAGE} \end{cases} \tag{5}$$

The formula to compute the decision score of a non-annotated vulnerability v is shown in Equation 6, assuming a knowledge base of k annotated vulnerabilities av_0 to av_{k-1}:

$$\text{score}(v) = \begin{cases} \dfrac{\sum_{i=0}^{k-1} \text{score}(av_i) \times \text{similarity}(v, av_i)}{\sum_{i=0}^{k-1} \text{similarity}(v, av_i)}, & \text{if } \exists i \in \{0, ..., k-1\} \quad \text{similarity}(v, av_i) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

Put another way, the alert decision score is the average decision score for all annotated vulnerabilities weighted by their similarity to the analyzed vulnerability, with a special case of returning 0 when there are no similar annotated vulnerabilities yet.

This formula implies that any vulnerability for which there are few or no similar annotated vulnerabilities gets an alert decision score close or equal to 0, resulting in a TICKET alert, in line with our hypothesis that this is the correct default reaction. Conversely, LOG and PAGE alerts are only emitted if the vulnerability is strongly similar to known vulnerabilities that were annotated as such.

3.5. Uncertainty Score

The active learning architecture we use [42] requires an *uncertainty metric* for each vulnerability, as part of the vulnerability selection process used to submit vulnerabilities to be annotated by the expert (which we discuss in Section 3.7). However they do not specify how to implement it.

We make the hypothesis that since TICKET is our default reaction, the closer to TICKET a vulnerability is, the more uncertain our decision is. Conversely, the closer to LOG or PAGE a decision is, the more certain our decision is. Therefore we compute uncertainty using Equation 7.

$$\text{uncertainty}(v) = 1 - |\text{score}(v)| \tag{7}$$

Therefore the uncertainty score of a vulnerability reaches 1 the closer its decision score is to 0 (TICKET), and reaches 0 the closer its decision score is to -1 or $+1$ (LOG or PAGE, respectively).

3.6. Similarity Matrix to Measure Information Density

We first describe as background the concepts of *similarity matrix* and *information density* proposed by Settles *et al* [42], as they are part of our architecture. We then present a slight modification to the scheme we add in the context of a dynamic dataset such as ongoing vulnerability disclosures.

3.6.1. Background: Similarity Matrix and Information Density

Assuming a set of n past vulnerabilities v_0 to v_{n-1}, a similarity matrix S of size $n \times n$ can be constructed using Equation 8.

$$\forall i, j \in \{0, ..., n-1\} \quad S_{i,j} = \text{similarity}(v_i, v_j) \tag{8}$$

From the similarity matrix S we compute an *information density* metric for every vulnerability by computing the average similarity of a vulnerability to every other vulnerabilities in the dataset, including itself (therefore no vulnerability can have an information density of zero). This is described by Equation 9.

$$\forall i \in \{0, ..., n-1\} \quad \text{density}(v_i) = \sum_{j=0}^{n-1} \frac{S_{i,j}}{n} \tag{9}$$

Information density is a measure of how *typical* a vulnerability is. When a lot of vulnerabilities resemble each other, they all have a high similarity metric when compared to each other, increasing the information density of each of them. Conversely, an outlier vulnerability with few or no similar vulnerabilities has a low information density.

3.6.2. Vulnerability Dataset Truncation

Settles *et al* proposed the concept of similarity matrix with a static dataset in mind. However, as many new vulnerabilities are disclosed every day, in our case the dataset is dynamic and the creation and update of a similarity matrix are quadratic through time and space. To limit the cost, we modify the scheme proposed by Settles *et al* and decide to truncate our vulnerability dataset once it goes over a certain size. We define a hyperparameter N for the maximum number of vulnerabilities we find acceptable, then remove excess vulnerabilities once $n > N$. For choosing the vulnerabilities to be truncated we simply remove the oldest vulnerabilities first.

3.7. Selecting a Vulnerability to be Evaluated

The main decision task of the training phase is to select the next vulnerability to be annotated by the expert. This raises several challenges. Selecting only the most typical vulnerabilities can lead to submit many vulnerabilities similar to each other, with diminishing returns after the first few ones. Conversely, selecting vulnerabilities only based on decision uncertainty leads to submit many outliers to the expert, which does not help with getting a broader understanding of the entire vulnerability set. Another question is how much the selection process should be based on randomness. A solely deterministic process could get the learning stuck in local

maxima of the vulnerability euclidean space, while leaning too much on randomness might lead to submit too many uninteresting vulnerabilities to the expert.

To address all these challenges, we propose an approach based on the work of Settles *et al* [42] which we present as background. We then propose a modification to incorporate randomness in the selection strategy.

3.7.1. Background: Selection Score and Selection Strategy

Settles *et al* [42] proposed that every vulnerability eligible for annotation should be given a *selection score* according to a *selection strategy*. They proposed several strategies for the selection score and we choose the simplest one as a starting point, shown in Equation 10. This strategy does not include randomness yet, which we describe in the next section.

$$\text{select}(v) = \text{density}(v) \times \text{uncertainty}(v) \tag{10}$$

As seen in Section 3.5, when a vulnerability has no similar annotated vulnerabilities, it gets an uncertainty score of 1. Therefore, at the beginning of the training (when there are no annotated vulnerabilities yet) all vulnerabilities get the same uncertainty score, and the selection process is based on information density only (we show below how we add randomness). A large group of vulnerabilities similar to each other (and therefore with high information density) is called a vulnerability *cluster*. Once more vulnerabilities get annotated inside the biggest clusters, the uncertainty score of the unannotated vulnerabilities in the clusters progressively decreases as there are similar annotated vulnerabilities in the knowledge base. At some point, other smaller clusters with lower information density but higher uncertainty (as they are unexplored yet) reach higher selection scores and get picked up first.

3.7.2. Adding Randomness to the Selection Process

With no randomness at all, the vulnerability selection is entirely deterministic, assuming a fixed set of vulnerabilities and an expert always choosing the same alert level for the same vulnerability. This can create problems as some vulnerability clusters are much denser than others, leading the selection process to waste too much of the expert's time on too few clusters. However, selecting a vulnerability from a smaller pool of randomly chosen vulnerabilities allows for more exploration of the vulnerability dataset as the denser clusters are not always present in the random pool. Therefore in our work we slightly modify the selection strategy proposed by Settles *et al* [42] to incorporate a certain amount of randomness by randomly selecting r vulnerabilities from the set of all candidates vulnerabilities to be annotated, before applying the rules described in Equation 10 to choose a vulnerability from this random subset. r is a hyperparameter. We claim that taken together, randomness, uncertainty, and information density provide the foundation for a robust annotation selection process.

Our complete training algorithm, including the vulnerability selection process is described in Algorithm 1.

```
Input: V: set of all past vulnerabilities
Input: B: annotation budget for the current training phase
Input: ONLINE: boolean set to true if the current training phase is online, false if offline
Input: r: size of the random subset of vulnerabilities
while B > 0 do
    C = get_all_unannotated_vulnerabilities(V);
    if ONLINE then
        C = retain_only_vulnerabilities_disclosed_in_last_period(C);
    end
    R = select_random_subset_of_vulnerabilities(C, r);
    v = select_vulnerability_with_highest_selection_score(R);
    submit_vulnerability_to_expert(v);
    B = B - 1;
end
```

Algorithm 1: Training algorithm, including the vulnerability selection process.

4. Preliminary Evaluation

Our evaluation goal is to check the real-world applicability of our prediction system, designed and configured with preliminary hypotheses, by confronting it to actual security experts in charge of real information systems. In our experimental protocol, these experts annotate vulnerabilities to assert their risk levels, in the context of the systems they are responsible for. The experiment is divided in two steps: in the first step, security experts train the prediction system, through an offline training followed by an online training, by simulating the passing of time. Second, they evaluate the prediction system: this is done by randomly selecting vulnerabilities, then submitting them to the prediction system and the expert simultaneously while comparing their answers. The main difference between the two steps is that during training the system controls which vulnerabilities are submitted to the expert, and the resulting annotations are added to the system's knowledge base. On the contrary during evaluation the vulnerability selection is completely random and the expert's annotations are only used as an evaluation tool without being added to the system's knowledge base.

A major challenge of this experiment is the requirement for busy security experts to commit a certain amount of time to participate, such as half a day. Moreover, as the training's vulnerability selection process simultaneously depends on the active learning architecture, its hyperparameters, and the expert previous choices, any iteration in the design or configuration of the prediction system requires to undertake a brand new experiment (including the expert full participation) to properly evaluate the changes.

In the end we have been able to complete two experiments, which we denote Experiment A and B. The information system studied in experiment A is a server-side software development environment, including a git server, an issue tracker, NTP and DNS servers, as well as Windows and Linux machines using a Kerberos-based authentication protocol. The information system studied during experiment B was also an information system for software development, with Linux machines only. It includes a git server, a Jenkins server, an issue tracker, a wiki system,

and is administered through SSH.

A shortcoming of this preliminary evaluation is that both experiments were done as part of a first experimental campaign, and we have not been able to secure the launch of a second experimental campaign based on what we learned during the first one. This campaign was based on initial hypotheses we took, some of which turned out to be questionable, resulting in mixed evaluation results. This is further discussed in Section 5.

We present our evaluation protocol in Section 4.1 and the results of the experiments in Section 4.2.

4.1. Experimental Protocol

Our experimental protocol uses past CVE vulnerabilities to simulate the production deployment of an alert system at a random past date. The protocol starts with an initial offline training session, then simulates the passing of time for an online training session, followed by an online evaluation session.

A major experimental constraint was that vulnerability annotations authored by our participants were actually sensitive themselves, as they provided a lot of insights into the related information systems. This required the whole campaign to be carried out without us ever seeing those annotations, or storing them on our own hardware. Therefore experiments were done exclusively on security experts' own machines (usually laptop workstations), preventing the use of dedicated server-grade computing resources. For this reason the vulnerability selection process, which is computationally intensive, had to be bounded in time and resources, as the experience had to stay interactive enough for the security expert to remain engaged while running entirely from a lightweight workstation. This is not straightforward as a similarity matrix for the entire CVE dataset requires hundreds of megabytes of storage, takes minutes to compute, and is only valid for a specific time step in the simulation as new vulnerabilities and keywords are added and discarded. We solved this engineering problem by storing all similarity matrices needed for every time step of the simulated timeline on the experts' workstations, ensuring the interactivity of the experience at the cost of each experiment requiring several hundreds of gigabytes of storage.

The experimental protocol was prepared in 2019, giving us access to all CVE vulnerabilities disclosed between 2007 and 2018. However, as explained in Section 3.6, the computation and storage of the vulnerability similarity matrix is quadratic in the number of vulnerabilities in our database, and differs at every time step (as more vulnerabilities are disclosed and more CPE URIs are added to the featurization word list). The hyperparameter N, first described in Section 3.6, is the maximum number of vulnerabilities the prediction system retains in its database. We empirically found that setting $N = 10000$ was close to the highest value we could set while keeping the user experience interactive using our current prototype. As we discard older vulnerabilities first and more than 5000 vulnerabilities have been disclosed every year for the last decade, in practice only vulnerabilities disclosed between 2014 and 2018 are actually used in the experiment.

Regarding the annotation budget, we had initial discussions with potential participants about the amount of time they would be able to commit to the experiments. From early participant feedback and preliminary empirical tests, we converged on an experimental duration of half a day

	System decision		
Expert decision	**LOG**	**TICKET**	**PAGE**
LOG	46	39	0
TICKET	4	2	0
PAGE	5	4	0

Table 1
Evaluation results for experiment A. Correct decisions: 48. False positives: 39. False negatives: 13.

	System decision		
Expert decision	**LOG**	**TICKET**	**PAGE**
LOG	35	36	6
TICKET	5	12	1
PAGE	2	3	0

Table 2
Evaluation results for experiment B. Correct decisions: 47. False positives: 45. False negatives: 10.

consisting in annotating 300 vulnerabilities, an amount to be divided into offline training, online training, and online evaluation. This budget was the best compromise between participant's availability and experimental validity.

Our first intuition was to divide the annotation budget evenly between offline training, online training, and online evaluation, providing a budget of 100 annotations for each step. For online training and evaluation, we also have to divide the budget into a number of weeks of simulated time and a number of vulnerabilities to be annotated per week. For online evaluation we settled on an even divide of 10 annotations per week during 10 weeks of simulated time. However early informal experiments highlighted a potential problem for online training: it is common for many vulnerabilities affecting the same software or hardware to be disclosed simultaneously, creating a risk of wasting part of the training budget as most vulnerabilities disclosed in a single period are affecting the same software. For this reason, we decided to spread the online training budget on more weeks, with only 8 annotations per week during 12 weeks of simulated time, for a total of 304 vulnerability annotations in the entire experiment.

The last hyperparameter to set was r, the number of randomly picked vulnerabilities to be ranked during the vulnerability selection process. Early informal experiments highlighted the risk of getting stuck into local maxima when carrying a mostly deterministic vulnerability selection process. Therefore we decided to let randomness have an important role in vulnerability selection by setting $r = 10$. We hypothesized that this setting would let the training process propose varied vulnerabilities to the expert, while still having a high probability of having at least one interesting vulnerability to submit to the expert among the ten random ones.

4.2. Results

Evaluation results for experiments A and B are shown in Table 1 and 2. Correct decisions are shown in the diagonal row. False positives (for which the system chose a higher alert level than the expert) are shown above the diagonal, while false negatives (for which the system chose a lower alert level than the expert) are shown below.

Results show that our approach can be improved: less than half of the decisions are identical between the system and the expert, with a very high number of false positives yet a significant amount of false negatives. The most common type of false positives is a vulnerability deemed benign by the expert (LOG, no alert raised) for which the system created a non-urgent TICKET for inspection during working hours. This is a consequence of the low base rate of non-LOG vulnerabilities (experts for experiment A and B respectively classified 85 % and 77 % of evaluation vulnerabilities as LOG) coupled with our design choice of favoring TICKET as a default response.

As we said in Section 4.1, we did not get access to the expert's annotations for neither experiment A or B. However, for experiment B we had the opportunity to follow up with the expert's team to get some insight into the evaluation process. An important discovery made during this follow-up is that at least five false negatives from experiment B are actually human errors instead of prediction errors (the five vulnerabilities annotated as PAGE by the expert were incorrectly annotated during the evaluation and the participant actually agrees with the system's decisions). According to the participant, this is partly due to a specific keyword in the vulnerability description that incorrectly startled him, and partly due to annotation fatigue, which we discuss in Section 5. We chose not to correct our results following this information for two reasons: first, there have been no comprehensive review of all evaluation vulnerabilities for neither experiments, leaving the possibility of even more undetected human errors. Moreover, we consider expert error a real-world condition that should be acknowledged by a robust evaluation protocol.

5. Discussion

As said previously, for both experiments A and B we did not get access to the 304 vulnerabilities submitted to the expert, nor the annotations provided by the expert, for confidentiality reasons. Under these conditions we can only speculate about what went right or wrong during the automated decision process, and can outline some design changes we would propose in the advent of another experimental campaign.

5.1. Experimental Limitations

It is likely that a budget of 304 vulnerabilities is not enough for both training and evaluating our prediction system. It is noteworthy that in the context of a production prediction system, having a security operator annotate vulnerabilities one hour a week for a year would result in an order of magnitude increase in annotation budget compared to our experimental campaign. We do not know how such an increase in budget would affect decision accuracy.

5.2. Decision Quality

For this work we only used dimensions generated by the keyword extraction pipeline described in our previous work [24], with past CPE URIs used as a filtering whitelist to prevent too many noisy keywords. This raises two risks: the quality of the extracted names may not be good enough for accurate decisions and comparing vulnerabilities by affected software alone may not be enough for proper risk analysis. As we could not study ourselves the training and

evaluation data, we delivered a debugging tool to experimenters allowing them to look back at any evaluation vulnerability. Using this tool they could list the training vulnerabilities that were considered similar, and the keywords from which the similarity arose. While they did not have time to do a comprehensive analysis of all evaluation decisions, they did report some valuable insights. Many decision mistakes were due to incorrect links between vulnerabilities, due to noisy keywords that are nevertheless present in the CPE URI whitelist: this is the case for common technical words such as "internet" (present in the whitelist because of many software names starting with *Internet Explorer*), "api" (because of software names such as *Broker API* or *API connect*), "credentials" (because of software names such as *Credential Bindings*). It could be worthwhile to improve the filtering capabilities of our keyword extraction pipeline, as well as giving security experts the opportunity to blacklist keywords that led to incorrect decisions in the past. Last but not least, it would be interesting to evaluate how adding other vulnerability properties would help risk analysis.

5.3. Expert Expectations

When explaining our experimental protocol to participants we did our best to convey that the current prediction system is only based on affected software names, and does not take into account other properties such as affected versions or vulnerability severity. However we never met directly the participant for experiment A (we presented the protocol to his colleagues) and more than a year had passed between initial project presentation and the starting of experiment B. This means participants may have forgotten (or were never aware to begin with) of this limitation while participating in the experiment, and participant's feedback following the experiments left this point unclear. If it was confirmed that security experts tend to include other information such as version or severity into their vulnerability analysis, this would be a strong indicator that more vulnerability properties are needed to increase prediction accuracy.

Another point where our system may have differed from participants' expectations is our choice to consider TICKET as a default choice. Our initial hypothesis was that false negatives are worse than false positives, as they imply the presence of an unmitigated risk for the information system. Many security experts we talked to disagreed with us: most of them told us that they had too many alerts from too many monitoring systems to handle them all, and they did not have time to check all newly disclosed CVE vulnerabilities manually. Therefore they strongly prefer some false negatives (which is not worse than their current situation) to false positives (which create more alerts for them to handle). A future version of this prediction system could allow the possibility for the security expert to choose a default alert behavior she feels appropriate.

5.4. Hyperparameters Tuning

Our learning system and evaluation protocol both require many hyperparameters, creating a hyperstate space too vast to be explored comprehensively. This is a problem shared by many machine learning endeavors, as changing a hyperparameter value requires the training to be started over from scratch. However active learning is especially vulnerable to this phenomenon as training requires the active participation of a human, and the choice of a sample to be annotated during training is impacted by the hyperparameters settings. As security experts'

time is expensive, any hyperparameter mistake is very costly. Many big and small decisions must be taken to prepare such an evaluation protocol, sometimes without the ability to justify these decisions appropriately. When the participating human is expected to be a domain-knowledge expert (as in our case), retraining the model is very expensive and doing it repeatedly is not possible in practice.

5.5. Decision Explicability

In our opinion, our lack of access to this campaign's experimental data validates our design choice to consider decision explicability as a critical property of a security decision system. This enables us to design self-serving debugging tools usable by participants, giving them the ability to understand by themselves the reasons why predictions succeeded or failed. The participants were able to give us all the valuable insights we discussed in this section while still keeping the sensitive details of their information system confidential. As discussed in Section 5.2, a natural next step would be to give the expert the tools to improve by themselves the quality of a decision after having diagnosed its root cause. A proposal that is simple for experts to understand and easy for us to implement would be to allow the expert to manually blacklist a low-quality keyword, for instance after having debugged an incorrect alert decision.

6. Conclusion

In this article we presented the initial design of an automated vulnerability risk analysis prediction system that can immediately choose an appropriate alert level for a just-disclosed vulnerability. It is interactively trained by a security operator using active learning: by mimicking the alert decisions of the human operator, the prediction system can gradually learn which vulnerabilities are considered important in the context of a specific information system. The system is based on an active learning architecture proposed by Settles *et al* [42] that includes the computation of a cosine similarity matrix and an information density vector. We modified this architecture to handle dynamic datasets such as the flow of vulnerability disclosures and to add a controllable amount of randomness in the vulnerability selection process. The architecture also uses our own work on vulnerability keyword extraction (described in our previous work [24]) as a featurization scheme.

This contribution is still at an early stage. Our experimental results show some of our initial design choices could be revisited. However, security experts we collaborated with expressed great interest in the concept and their collaboration was a crucial aspect of this contribution. Notably, this partnership highlighted how decision explicability is an important factor for real-world applicability of machine learning for security alerting. Going forward we would like to carry on this relationship onward as well as extend it to include even more participants, opening the door to a more iterative research process and more sound experimental protocols. Validating such a system would be an important step forward in the defense against newly disclosed vulnerabilities, allowing organizations to set up an around-the-clock vulnerability monitoring system while only requiring their security operators to work during regular hours to train the system.

References

[1] L. Bilge, T. Dumitraş, 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), ACM, New York, NY, USA, 2012, pp. 833–844. doi:10.1145/2382196.2382284.

[2] US10616258B2 - Security information and event management, 2020-01-09. URL: https://patents.google.com/patent/US10616258B2/en.

[3] Common Vulnerabilities and Exposures (CVE), 2020-08-04. URL: https://cve.mitre.org/.

[4] National Vulnerability Database, 2020-08-04. URL: https://nvd.nist.gov/.

[5] Security Content Automation Protocol, 2020-08-04. URL: https://csrc.nist.gov/projects/security-content-automation-protocol.

[6] S. Frei, M. May, U. Fiedler, B. Plattner, Large-scale Vulnerability Analysis, in: Proceedings of the 2006 SIGCOMM Workshop on Large-scale Attack Defense (LSAD '06), ACM, New York, NY, USA, 2006, pp. 131–138. doi:10.1145/1162666.1162671.

[7] S. Clark, S. Frei, M. Blaze, J. Smith, 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), ACM, New York, NY, USA, 2010, pp. 251–260. doi:10.1145/1920261.1920299.

[8] M. Shahzad, M. Z. Shafiq, A. X. Liu, A Large Scale Exploratory Analysis of Software Vulnerability Life Cycles, in: Proceedings of the 34th International Conference on Software Engineering, IEEE Press, Piscataway, NJ, USA, 2012, pp. 771–781. URL: http://dl.acm.org/citation.cfm?id=2337223.2337314.

[9] S. Frei, D. Schatzmann, B. Plattner, B. Trammell, Modeling the security ecosystem - the dynamics of
(in)security, in: T. Moore, D. Pym, C. Ioannidis (Eds.), Economics of Information Security and Privacy, Springer US, Boston, MA, 2010, pp. 79–106.

[10] K. Nayak, D. Marino, P. Efstathopoulos, T. Dumitraş, Some vulnerabilities are different than others, in: A. Stavrou, H. Bos, G. Portokalidis (Eds.), Research in Attacks, Intrusions and Defenses, Springer International Publishing, Cham, 2014, pp. 426–446.

[11] L. Allodi, Economic factors of vulnerability trade and exploitation, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17, Association for Computing Machinery, New York, NY, USA, 2017, p. 1483–1499. URL: https://doi.org/10.1145/3133956.3133960. doi:10.1145/3133956.3133960.

[12] C. Sabottke, O. Suciu, T. Dumitras, Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits, in: 24th USENIX Security Symposium (USENIX Security 15), USENIX Association, Washington, D.C., 2015, pp. 1041–1056. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/sabottke.

[13] Common Vulnerability Scoring System, 2020-08-04. URL: https://www.first.org/cvss/.

[14] S. H. Houmb, V. N. Franqueira, E. A. Engum, Quantifying security risk level from cvss estimates of frequency and impact, Journal of Systems and Software 83 (2010) 1622 – 1634. URL: http://www.sciencedirect.com/science/article/pii/S0164121209002155. doi:https://doi.org/10.1016/j.jss.2009.08.023, software Dependability.

[15] L. Allodi, F. Massacci, A preliminary analysis of vulnerability scores for attacks in wild: The ekits and sym datasets, in: Proceedings of the 2012 ACM Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, BADGERS '12, Association for Computing Machinery, New York, NY, USA, 2012, p. 17–24. URL: https://doi.org/10.1145/2382416.2382427. doi:10.1145/2382416.2382427.

[16] L. Allodi, F. Massacci, Comparing vulnerability severity and exploits using case-control studies, ACM Trans. Inf. Syst. Secur. 17 (2014). URL: https://doi.org/10.1145/2630069. doi:10.1145/2630069.

[17] OSVDB Shut Down Permanently - Securityweek.com, 2020-28-11. URL: https://www.securityweek.com/osvdb-shut-down-permanently.

[18] M. Bozorgi, L. K. Saul, S. Savage, G. 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, KDD '10, Association for Computing Machinery, New York, NY, USA, 2010, p. 105–114. URL: https://doi.org/10.1145/1835804.1835821. doi:10.1145/1835804.1835821.

[19] S. Mittal, P. K. Das, V. Mulwad, A. Joshi, T. Finin, Cybertwitter: Using twitter to generate alerts for cybersecurity threats and vulnerabilities, in: 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), 2016, pp. 860–867.

[20] N. Tavabi, P. Goyal, M. Almukaynizi, P. Shakarian, K. Lerman, Darkembed: Exploit prediction with neural language models, 2018. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17304.

[21] J. Jacobs, S. Romanosky, B. Edwards, M. Roytman, I. Adjerid, Exploit Prediction Scoring System (EPSS), in: Black Hat 2019, 2019. URL: http://i.blackhat.com/USA-19/Thursday/us-19-Roytman-Predictive-Vulnerability-Scoring-System-wp.pdf.

[22] David A. Wheeler - Shellshock, 2020-03-08. URL: https://dwheeler.com/essays/shellshock.html.

[23] C. Elbaz, Reacting to "n-day" vulnerabilities in information systems, Theses, Université Rennes 1, 2021. URL: https://tel.archives-ouvertes.fr/tel-03350455.

[24] C. Elbaz, L. Rilling, C. Morin, Automated keyword extraction from "one-day" vulnerabilities at disclosure, in: NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium, 2020, pp. 1–9.

[25] C. Elbaz, L. Rilling, C. Morin, Fighting n-day vulnerabilities with automated cvss vector prediction at disclosure, in: Proceedings of the 15th International Conference on Availability, Reliability and Security, ARES '20, Association for Computing Machinery, New York, NY, USA, 2020. URL: https://doi.org/10.1145/3407023.3407038. doi:10.1145/3407023.3407038.

[26] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, Y. Yarom, Spectre attacks: Exploiting speculative execution, in: 2019 IEEE Symposium on Security and Privacy (SP), 2019, pp. 1–19.

[27] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, M. Hamburg, Meltdown, 2018. arXiv:1801.01207.

[28] NVD - CPE, 2020-08-04. URL: https://nvd.nist.gov/products/cpe.

[29] The Myth of Software and Hardware Vulnerability Management, 2016-05-04. URL: https://www.foo.be/2016/05/The_Myth_of_Vulnerability_Management/.

[30] Debian – Security Information, 2020-12-11. URL: https://www.debian.org/security/.
[31] The 2019 Official Annual Cybersecurity Jobs Report, 2020-20-07. URL: https://www.herjavecgroup.com/2019-cybersecurity-jobs-report-cybersecurity-ventures/.
[32] The Mad Dash to Find a Cybersecurity Force - The New York Times, 2020-20-07. URL: https://www.nytimes.com/2018/11/07/business/the-mad-dash-to-find-a-cybersecurity-force.html.
[33] J. Dykstra, C. L. Paul, Cyber operations stress survey (coss): Studying fatigue, frustration, and cognitive workload in cybersecurity operations, in: 11th USENIX Workshop on Cyber Security Experimentation and Test (CSET 18), USENIX Association, Baltimore, MD, 2018. URL: https://www.usenix.org/conference/cset18/presentation/dykstra.
[34] S. C. Sundaramurthy, J. McHugh, X. S. Ou, S. R. Rajagopalan, M. Wesch, An anthropological approach to studying csirts, IEEE Security Privacy 12 (2014) 52–60.
[35] S. C. Sundaramurthy, A. G. Bardas, J. Case, X. Ou, M. Wesch, J. McHugh, S. R. Rajagopalan, A human capital model for mitigating security analyst burnout, in: Eleventh Symposium On Usable Privacy and Security (SOUPS 2015), USENIX Association, Ottawa, 2015, pp. 347–359. URL: https://www.usenix.org/conference/soups2015/proceedings/presentation/sundaramurthy.
[36] O. Ogbanufe, J. Spears, Burnout in cybersecurity professionals, WISP 2019 - Workshop in Information Security and Privacy, Munich, Germany, 2019.
[37] W. Chappelle, K. McDonald, J. Christensen, L. Prince, T. Goodman, W. Thompson, W. Hayes, Sources of Occupational Stress and Prevalence of Burnout and Clinical Distress Among U.S. Air Force Cyber Warfare Operators, Technical Report, School of Aerospace Medicine Wright Patterson AFB OH, 2013.
[38] B. Settles, Active learning literature survey, Technical Report, University of Wisconsin-Madison Department of Computer Sciences, 2009.
[39] A. Beaugnon, P. Chifflier, F. Bach, Ilab: An interactive labelling strategy for intrusion detection, in: International Symposium on Research in Attacks, Intrusions, and Defenses, Springer, 2017, pp. 120–140.
[40] K. S. Jones, A statistical interpretation of term specificity and its application in retrieval, Journal of Documentation 28 (1972) 11–21.
[41] Term Frequency - Inverse Document Frequency statistics, 2020-12-11. URL: https://jmotif.github.io/sax-vsm_site/morea/algorithm/TFIDF.html.
[42] B. Settles, M. Craven, An analysis of active learning strategies for sequence labeling tasks, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP '08, Association for Computational Linguistics, USA, 2008, p. 1070–1079.
[43] B. Beyer, C. Jones, J. Petoff, N. R. Murphy, Site Reliability Engineering: How Google Runs Production Systems, " O'Reilly Media, Inc.", 2016.