A Recommender System for User-Specific Vulnerability Scoring

(full version)

Linus Karlsson, Pegah Nikbakht Bideh, Martin Hell

Lund University, Department of Electrical and Information Technology, Sweden
{linus.karlsson, pegah.nikbakht_bideh, martin.hell}@eit.lth.se

Abstract. With the inclusion of external software components in their software, vendors also need to identify and evaluate vulnerabilities in the components they use. A growing number of external components makes this process more time-consuming, as vendors need to evaluate the severity and applicability of published vulnerabilities. The CVSS score is used to rank the severity of a vulnerability, but in its simplest form, it fails to take user properties into account. The CVSS also defines an environmental metric, allowing organizations to manually define individual impact requirements. However, it is limited to explicitly defined user information and only a subset of vulnerability properties is used in the metric. In this paper we address these shortcomings by presenting a recommender system specifically targeting software vulnerabilities. The recommender considers both user history, explicit user properties, and domain based knowledge. It provides a utility metric for each vulnerability, targeting the specific organization's requirements and needs. An initial evaluation with industry participants shows that the recommender can generate a metric closer to the users' reference rankings, based on predictive and rank accuracy metrics, compared to using CVSS environmental score.

1 Introduction

The current software development landscape shows a trend towards increasing reuse of existing code. Products are constructed by using already existing libraries and software, such as OpenSSL, libxml, and many others. A report [20], found that in the scanned applications, on average 57% of the code base was open source. However, as a maintainer of products, vendors need to identify vulnerabilities in the components they use. As the number of external components increases, the workload on developers to identify vulnerabilities and update these components grows. At the same time, many vendors already have a hard time to identify and evaluate vulnerabilities, for example in IoT companies [8].

Updating a component introduces a cost, since it requires a new release cycle to be completed. This includes building, quality assurance, and the distribution of the new release to the end-users' devices. Therefore, vendors would like to patch only vulnerabilities which are relevant to the product.

The Common Vulnerability Scoring System (CVSS) [5, 11] defines a severity ranking for vulnerabilities. The base score does not take into account individual preferences of users. Instead, CVSS has an environmental metric which can be used to modify the base score such that it represents user dependent properties of vulnerabilities. It will rewrite the confidentiality, integrity, and availability metrics both to adjust them according to measures already taken by the organization, but also to capture the actual impact such loss would have on the organization. As this will differ between organizations, such a modified metric will better reflect the actual severity of a vulnerability to that organization.

The environmental metrics must be evaluated on a per vulnerability basis and are handled manually. This is both time consuming, error prone, and can lead to inconsistencies in case there are several vulnerabilities and they are handled by different analysts. Moreover, the environmental metric, though unique for the organization, only constitutes the sub-metrics available in the base score. Additional information that might affect the organization is not covered.

Recommender systems work by analyzing information about user preferences, and combine this with information about items, or with the history of other users. Their goal is to output recommendations targeting the specific user.

In this paper, we explore ways to improve measuring how a vulnerability affects an organization. Using machine learning techniques applied to recommender systems, we combine different properties and metrics in order to capture vulnerability data and map it to requirements of the specific organization. Compared to CVSS environmental metrics, our method provides several advantages.

First, the requirements for the organization is derived by combining explicit requirements with requirements learned from previous analysis of vulnerabilities. This data driven approach will not only use personal preferences, but also take into account how real vulnerabilities have been evaluated previously. Such learned data is able to capture information that might be overseen by analysts, or that are difficult to express. Second, our approach is general and is not restricted to a certain group of properties. It can be amended with new metrics if needed, focusing on metrics relevant for the given organization or device.

Our goal is to design a recommender that provides a personalized severity assessment based on a user profile. The profile is both explicit, based on the users' own choices, and implicit as the recommender learns from the users' previous actions. We also support inclusion of domain knowledge into the system and discuss how the different parts can be weighted, following a heuristic approach. Suitable similarity functions are used to form a utility function that outputs the personalized severity assessment. The recommender is also evaluated using participants from the industry. Though the evaluation is small scale, the results indicate that our recommender system is able to provide severity information that is closer to the users' actual preferences than the CVSS environmental score.

The remainder of this paper is structured as follows: in Section 2 we describe the necessary background of recommender systems and vulnerabilities. In Section 3 the proposed model is described, which is followed by the implementation

of the model in Section 4. The recommender is evaluated in Section 5. Related work is discussed in Section 6. Finally, the paper is concluded in Section 7.

2 Recommenders and Vulnerability Severity Ratings

Generally, the goal of a recommender is to present recommendations of *items* to a set of *users*. An item can be for example a movie, a song, or a website. The idea is that the recommender should present a subset of items to the user, such that the user finds this subset relevant. The subset is found by matching user preferences or activity using a learnt profile and sometimes other similar users' activity. In a shopping scenario, the added value for the user also leads to higher sales. In this paper, the goal of the recommender is to add value to an end-user by tailoring the severity score for vulnerabilities.

Recommender systems can be divided into three major categories [1]: knowledge-based systems, content-based systems, and collaborative filtering.

A knowledge-based recommender system can be used in cases where ratings of items are not available, e.g., rarely used or bought items. It finds similarities between user requirements and item descriptions. In other words, a knowledge-based recommender allow users to specify desired domain-specific properties of items, and the recommender tries to find suitable items.

In content-based systems, item descriptions are used for recommendations and user ratings are combined with item information. One advantage of content-based systems is that when a rating is not available for an item, items with similar attributes that have been rated by the user can be used to make recommendations. On the other hand, because the lacking history of ratings for new users, they are not effective at providing recommendations for new users.

Collaborative filtering systems use collaborative ratings provided by multiple users to make recommendations. If two users have similar taste of ratings for many items, this similarity is identified. When only one of them has specified a rating, the other user can receive a similar rating.

Other than the types above, recommendations can be generated from domain-specific knowledge. This generates recommendations for a specific field of knowledge, and is designed specifically to handle data for that domain.

The above recommenders works well in defined scenarios. Knowledge-based systems are efficient in cold-start settings, while collaborative methods works well when a lot of ratings are available. Various features of different recommenders can be combined in hybrid systems for better performance.

Many vulnerabilities are reported and given a CVE identifier. The CVE system thus provides a centralized repository for vulnerabilities. As an example, in 2018, more than 16,500 vulnerabilities were added to the National Vulnerability Database (NVD). For each vulnerability, NVD also provides a severity score. This score, denoted the base score, uses exploitability and impact submetrics in order to define a severity score between 0–10. This score is made to be reproducible and organization independent. Instead, the environmental score can be used to adapt the base score to an organization's requirements and needs. In this

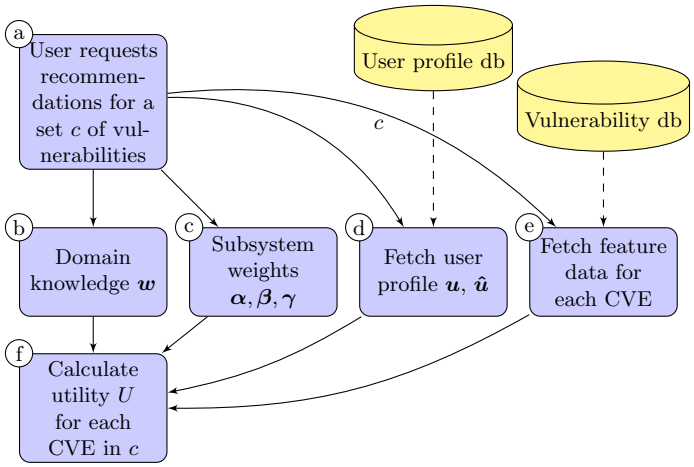

Fig. 1. Flow chart of recommendation generation

paper, a recommender system has been applied to CVEs, and the performance is compared to the environmental metric.

3 System Model

During the design of a recommender system for vulnerabilities, several requirements should be fulfilled. The following requirements have been identified: 1) The recommender should give reasonable recommendations for new users of the system, and thus avoid the cold-start problem of recommender systems. 2) It should allow the user to select certain preferences that the system will honor. 3) It should expose a meaningful subset of user preferences to the user. 4) It should learn from user actions, so that future recommendations are as relevant as possible to the user. To avoid privacy concerns, only the user's own actions are considered. Thus, methods based on collaborative filtering will not be considered in this paper.

We first note that no single class of recommender system can fulfill all requirements. Instead, we propose a hybrid recommender based on three parts. The first is a *domain-based* subsystem which provides domain-specific knowledge unique to a recommender for vulnerabilities. The second part is a *knowledge-based* subsystem which allows the user to select certain user preferences that they are interested in. Lastly, the third part is a *content-based* subsystem which learns from the user's previous actions to provide more meaningful recommendations for each user.

3.1 Overall Recommender System Design

An overview of the recommendation generation process can be seen in Fig. 1. When a user requests recommendations for a set c of vulnerabilities, the feature

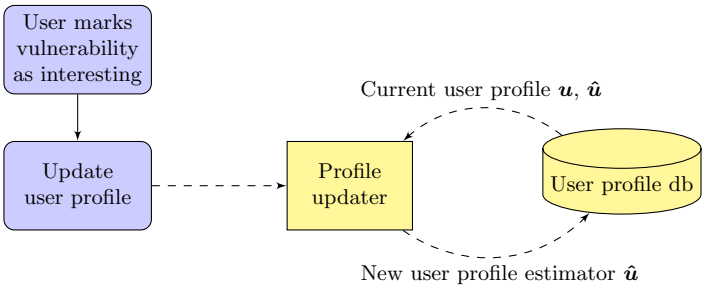

Fig. 2. Flow chart of user rating a vulnerability

data, domain-specific knowledge, user profiles, and weights are fetched from their respective storage. Each of these parts will be described in details in the following sections. These pieces will then be combined in the actual recommender, which then outputs recommendations in the range $[0, 1]$. Such a value is generated for each vulnerability in the set c of user requested vulnerabilities. A higher value means that a vulnerability is more relevant to the user.

Our hybrid recommender system learns user preferences based on the user's interaction with vulnerabilities. An overview of the rating procedure is shown in Fig. 2. First, the user rates a vulnerability based on their own preferences. While such a rating can be of any form, this paper only considers positive feedback. Next, the current user profile is updated with the new information, so that a new user profile estimated called $\hat{\boldsymbol{u}}$ is stored in the user profile database. The profile update procedure will be explained in Section 3.8.

3.2 Feature Representation

A key task in designing a recommender is constructing a good feature extraction stage. In our case, this means that we wish to extract features from each vulnerability, to be used as input to the recommender, see block (e) in Fig. 1. First, a selection of features must be made, and later on their respective feature weight parameters must be decided. We will discuss actual features to use in Section 4.1, while here we describe how features are represented inside the recommender.

We consider the features of a vulnerability as a vector $\boldsymbol{v}$, where each individual feature v_i denotes a specific feature value. Such a value could be of any type, such as a Boolean value, a real number, an integer in a specific range, categorical data, or hierarchical data.

3.3 User Profile Representation

There are two distinct parts of the user profile. First, there is the explicit user profile $\boldsymbol{u}$, where the user explicitly select their own preferences. This is similar to the requirements that can be defined in the CVSS environmental metric. Second, there is the *estimated* user profile $\hat{\boldsymbol{u}}$, which is determined from the

user's interactions with the system. The system learns this profile about the user automatically. This allows the system to capture user preferences that are hard to explicitly express for users, either because the feature is complex, or because the user is unaware of them. The explicit user profile is the knowledge-based part of our hybrid recommender, while the estimated user profile is the content-based part.

Each of the two parts of the user profile is represented as a vector, where each element of the vector describes the interest the user has for each feature. The elements of the vectors are matched with the feature value from above, to find vulnerabilities to recommend to the user. This matching is done by a *similarity* function, which is further discussed in Section 3.6. In Section 3.8 we describe how the estimated user profile vector is found.

3.4 Domain-specific Knowledge

The recommendations are not only based on the user profile, but also on a set of domain-specific knowledge, unique to the field of vulnerability assessment. Such knowledge is required both to provide recommendations suitable for such a highly specific area of interest, but also serves as a component to solve the cold-start problem.

The domain-specific knowledge $\boldsymbol{w}$ is represented in the same way as the user profile above, but instead of being user-specific, it is global for all users of the system. It is fetched at point (b) in Fig. 1. It can be used to express rules that should apply for all users, such as prioritizing recent vulnerabilities, or prioritizing vulnerabilities with lots of activity on social media.

3.5 Subsystem Weights

As described earlier, the recommender system is a hybrid system with three major parts. The three parts all contribute to the final result of the recommender, but they should be able to do so to different extents depending on the features, see Section 4.1. The subsystem weights are fetched at point (c) in Fig. 1.

The subsystems are given a weight between 0 and 1. Let the vectors $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$ describe the weights for the domain-based, knowledge-based, and content-based subsystems, respectively. For any given feature i, the sum $\alpha_i + \beta_i + \gamma_i = 1$. Note that relative weight of each subsystem can vary between different features.

3.6 Similarity Functions

A similarity function compares a value from the user profile, called the target value t_i, with the feature value extracted from the vulnerability v_i. We denote this function $\mathsf{sim}_i(t_i, v_i)$, where $0 \leq \mathsf{sim}_i(t_i, v_i) \leq 1$. Higher value means that the feature value is more similar to the target value. Here, we use the similarity functions given below. For examples of other variants, see e.g. [18].

The similarity function for the distance between t_i and v_i is given by

$$\mathsf{sim}_{\text{dist}}(t_i, v_i) = 1 - \frac{|t_i - v_i|}{\max_{\text{dist}} - \min_{\text{dist}}}, \tag{1}$$

where $\max_{\text{dist}}$ and $\min_{\text{dist}}$ are the maximum and minimum possible distances between t_i and y_i. This guarantees that the output is in the range $[0, 1]$.

Another similarity function is a scoring function, which sees the target value t_i as a multiplier to multiply the feature value with. This is suitable when we simply wish to rank higher feature values higher.

$$\mathsf{sim}_{\text{mult}}(t_i, v_i) = t_i \cdot v_i \tag{2}$$

Note that t_i must be selected so that the output range still stays within $[0, 1]$.

In the two previous similarity functions, both the target value t_i and the feature value v_i have been numerical values. However, as described earlier in Section 3.2, they can be of any type. Two examples of such similarity functions are $\mathsf{sim}_{\text{daydist}}$ and $\mathsf{sim}_{\text{cosine}}$, which calculates the difference between two dates, and the cosine similarity between two vectors, respectively. The date similarity $\mathsf{sim}_{\text{daydist}}$ can be implemented as in (1), with the date being days since the epoch, while the cosine similarity is calculated using

$$\mathsf{sim}_{\text{cosine}}(t_i, v_i) = \sum_{i=1}^{n} t_{ij} v_{ij} \Bigg/ \sqrt{\sum_{i=1}^{n} t_{ij}^2} \sqrt{\sum_{i=1}^{n} v_{ij}^2}, \tag{3}$$

where t_{ij} and v_{ij} are the j^{th} components of the vector t_i and v_i respectively.

A special case is a similarity function for Boolean values. In this case, t_i is simply a constant which is returned if v_i is true.

$$\mathsf{sim}_{\text{boost}}(t_i, v_i) = \begin{cases} t_i, & \text{if } v_i \text{ is true,} \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

Note that the similarity functions as described above follows the definition from [18], where the similarity function compares individual feature values.

3.7 Generating Recommendations

Combining the building blocks from the sections above, a complete recommender can now be described. The goal here is to describe a *utility function* U, which takes a given vulnerability $\boldsymbol{v}$ as input, and outputs the utility value, i.e. the user-specific severity assessment. As can be seen at point (f) in Fig. 1, the utility function U is the final step in a series of actions.

Recall that the design is a hybrid recommender. Therefore, subsystem weights will be combined with the similarity functions for the different feature values for all d features. Utility U for vulnerability can be described as:

$$U = \frac{1}{d} \sum_{i=1}^{d} \alpha_i \cdot \mathsf{sim}_i(w_i, v_i) + \beta_i \cdot \mathsf{sim}_i(u_i, v_i) + \gamma_i \cdot \mathsf{sim}_i(\hat{u}_i, v_i), \tag{5}$$

where $\alpha_i, \beta_i, \gamma_i$ are the subsystem coefficients, sim_i is the similarity function for the i^{th} feature, $w_i, u_i, \hat{u}_i$ are the target values for feature i for the different subsystems (i.e. elements of $\boldsymbol{w}, \boldsymbol{u}, \hat{\boldsymbol{u}}$ respectively), and v_i is the feature value for feature i.

Because the similarity functions are limited to the range $[0, 1]$, and $\alpha_i + \beta_i + \gamma_i = 1$, the output of U will be a value between 0 and 1. A higher value indicates higher utility, i.e. a better match to the user's preferences.

3.8 Updating User Profile

For estimating the user profile $\hat{\boldsymbol{u}}$, we wish to combine the previous estimation with the new data about the user's preferences. We consider only input of vulnerabilities that the user *is interested in*, that is, positive training examples. Then, the update function update can be expressed as a function of the form $\hat{\boldsymbol{u}}' = \mathsf{update}(\hat{\boldsymbol{u}}, \boldsymbol{v})$, i.e., a function taking a new vulnerability $\boldsymbol{v}$, the current $\hat{\boldsymbol{u}}$, and returning a new estimation of the user preferences $\hat{\boldsymbol{u}}'$.

Depending on what kind of user preferences the system should model, there are different ways to design the update function. In [12] the authors used the vector space model to represent text from web pages. The user profile was represented as a single vector $\hat{\boldsymbol{u}}$, therefore the authors represented their update function as $\hat{\boldsymbol{u}}' = a \cdot \hat{\boldsymbol{u}} + \boldsymbol{v}$, where a is a decay factor. Since the vectors had weights determined by the tf-idf scheme, in combination with using the cosine similarity measure, simple addition of the vectors worked well as an update function, because the cosine similarity measures vector orientation, not magnitude.

We propose an approach inspired by the paper above, with some adaptions to make the update function applicable for any type of feature, not only text. The proposed update function is given by

$$\mathsf{update}(\hat{\boldsymbol{u}}, \boldsymbol{v}) = (\mathsf{mer}_1(\hat{u}_1, v_1), \ldots, \mathsf{mer}_i(\hat{u}_i, v_i), \ldots, \mathsf{mer}_d(\hat{u}_d, v_d)) \ , \tag{6}$$

where d is the number of features, and therefore elements in $\hat{\boldsymbol{u}}$ and $\boldsymbol{v}$.

For each pair $(\hat{u}_i, v_i)$, a *merge function* mer_i is applied. The merge function is similar to the similarity functions sim_i, but instead of comparing two elements, it merges them. The merging needs to be handled different for each feature type, and this construction is thus a generalization of [2,12], where the merge function is equivalent to $\mathsf{mer}_i(\hat{u}_i, v_i) = \hat{u}_i + v_i$.

Another example of a more complex merge function, used later in this paper, is a merge function based on the Modified Moving Average (MMA):

$$\mathsf{mer}_{\text{mma}}(\hat{u}_i, v_i) = \frac{(S-1)\hat{u}_i + v_i}{S} \ , \tag{7}$$

where S controls the exponential smoothing.

Just as in Section 3.6, where similarity functions could handle both scalars and vectors, depending on the feature type, merge functions must support this as well. Consider the case where $\hat{u}_i$ and v_i are vectors, then the following merge function performs element-wise addition over n-dimensional vectors $\hat{u}_i$ and v_i.

$$\mathsf{mer}_{\text{add}}(\hat{u}_i, v_i) = (\hat{u}_{i,1} + v_{i,1}, \hat{u}_{i,2} + v_{i,2}, \ldots, \hat{u}_{i,n} + v_{i,n}) \ , \tag{8}$$

4 Implementation

Given the theoretical model described in the previous section, the actual recommender can now be constructed. At first, the set of features must be selected. While the proposed model is flexible enough to handle many different feature types, an actual implementation must still have suitable similarity functions, merge functions, and subsystem weights for all selected features. This section describes such decisions for our implemented recommender. We stress that this is an example implementation of the model described in the previous section. Another implementation may choose different features, weights, or functions.

4.1 CVE Features

The implementation has used several sources for vulnerability information. A majority of the data is collected from NVD [15], but also other sites such as CVEdetails [13], and Google have been used. A list of features extracted is available in Table 1, and below we discuss the features in more detail.

Table 1. Feature selection in the implementation, feature types, weights of domain-based (α), knowledge-based (β), and content-based (γ) subsystems, and finally similarity and merge functions

		Subsystem weights			Functions	
Features	Data type	α	β	γ	sim	mer
Impact metrics	Categorical	0.0	0.5	0.5	$\mathsf{sim}_{\text{mult}}$	$\mathsf{mer}_{\text{mma}}$
Exploitability subscore	Numerical	0.0	0.8	0.2	$\mathsf{sim}_{\text{mult}}$	$\mathsf{mer}_{\text{mma}}$
Authentication	Categorical	0.3	0.35	0.35	$\mathsf{sim}_{\text{mult}}$	$\mathsf{mer}_{\text{mma}}$
Access vector	Categorical	0.3	0.35	0.35	$\mathsf{sim}_{\text{dist}}$	$\mathsf{mer}_{\text{mma}}$
CWE	Hierarchical	0.0	0.0	1.0	$\mathsf{sim}_{\text{cosine}}$	$\mathsf{mer}_{\text{add}}$
Published date	Date	1.0	0.0	0.0	$\mathsf{sim}_{\text{daydist}}$	N/A
Metaspolit exploits	Boolean	0.3	0.7	0.0	$\mathsf{sim}_{\text{boost}}$	N/A
Linked external resources	Numerical	1.0	0.0	0.0	$\mathsf{sim}_{\text{mult}}$	N/A
Google hits	Numerical	1.0	0.0	0.0	$\mathsf{sim}_{\text{mult}}$	N/A

Impact metrics includes the impact metrics in the CVSS score, namely confidentiality, integrity, and availability impact. These are categorical values where the impact can be `NONE`, `PARTIAL`, or `COMPLETE`. In our implementation, we map these values to numerical scores of 0.0, 0.5, and 1.0 respectively. These metrics are interesting since they describe how serious the impact is on different security properties.

Exploitability subscore is the exploitability subscore from the CVSS ranking, which estimates the ease of exploiting the vulnerability. This is a numerical value between 0.0 and 10.0. This metric is interesting since a higher value means that the vulnerability is easier to exploit.

Authentication (CVSS metric) describes how many times an attacker needs to authenticate before performing an attack. It is a categorical feature with values `NONE`, `SINGLE`, or `MULTIPLE`. In our implementation, we map these values to numerical scores of 1.0, 0.5, and 0.0 respectively. This is interesting since if no authentication is required, a vulnerability is easier to exploit.

Access vector (CVSS metric) describes the attack vector for the vulnerability. It is categorical with the value `NETWORK`, `ADJACENT`, `LOCAL`. In our implementation, we map these to numerical values of 1.0, 0.5, and 0.0, respectively. The access vector is relevant since different users have different threat models. Some users may consider network attacks as most serious, while others may perceive local attacks as more serious. Later in the paper we will describe how the recommender allows the user to describe their preferences.

CWE ID categorizes vulnerabilities according to the type of the vulnerability. This is a hierarchical structure, but we treat each individual CWE as a category in our implementation, because there is only a limited set of all possible CWE IDs that are actually used in CVEs. We expect the CWE ID to be important in providing recommendations based on the user's history, since it describes the vulnerability class.

Metasploit exploits is a Boolean value which describes if there is a Metasploit module [16] available for this vulnerability. A module in Metasploit means that attackers may find and launch attacks through easy-to-use tools. Thus, such vulnerabilities are considered more serious.

Linked external resources is a numerical value which counts the number of linked resources for a specific CVE on NVD. These resources may include links to news articles, exploits, or security advisories. A vulnerability with a high amount of external resources may be more relevant to consider.

Google hits is a numerical value of the number of Google search hits a specific CVE-ID has. Just like the number of linked external resources, this tells the recommender about the popularity of the vulnerability in the Internet.

4.2 User Requirements Selection

When users start using the system, they should select what makes certain vulnerabilities more relevant to them. This is used to create the explicit user profile $\boldsymbol{u}$ for the recommender. The user profile is constructed by rating the importance of certain information about a vulnerability. The rating should be in the interval of [0,1], and will be used to construct the vector $\boldsymbol{u}$. User requirements can be selected in many ways, in our implementation the user can rate the following properties.

- Confidentiality impact: To what extent the vulnerability may cause private information to be leaked to an attacker.
- Integrity impact: To what extent the vulnerability may cause stored information to be modified by an attacker.
- Availability impact: To what extent the vulnerability may cause a system to be unavailable to perform its normal functions.

– Exploit accessibility: How widely accessible or easily used attack code that can be found for the vulnerability.
– Access vector: What access vector that is used for the attack, e.g. if network access is enough, or if local access is required.
– Authentication: If the vulnerability requires an already authenticated user, of if unauthenticated users can trigger the vulnerability as well.

4.3 Subsystem Weights

The choice of subsystem weights for each individual feature is based on two main aspects: *(i)* is it meaningful for users to explicitly state their preference about the feature, and *(ii)* do users' preferences for the feature differ, or do all users value the feature in the same way?

Recall that $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$ correspond to the domain-based, knowledge-based, and content-based parts of the recommender, respectively. In Table 1 the choice of subsystem weights for each feature can be seen. We discuss our choices below, but other choices are of course possible.

Since impact metrics are highly user-dependent, the domain-based part is set to 0.0, so that the user's explicit choice and history are the only things affecting the score for the impact metrics. In addition, an even split between explicit and implicit user preferences is selected. Similar arguments can be made for the Exploitability subscore, but since we wish the users to have a higher degree on explicitly selecting the importance of this setting, we have a larger β for this case.

The Authentication and Access vector features have a non-zero domain-based component, since the importance of these factors can be considered more universal across users.

Looking at the CWE ID, the type of underlying weakness that is of interest can be very user-dependent. Since there are several available CWE IDs, the recommender solely learns the user profile based on the user's previous action. Thus, both the domain-based and knowledge-based components are zero.

The opposite is true for the publication date, which is treated equally for all users, thus marking more recent vulnerabilities as more important. The same argument can be made for the number of linked external resources, and the number of Google hits.

Finally, the availability of Metasploit exploits is seen as a combination of a domain-based and knowledge-based preference.

4.4 Similarity and Merge Functions

The choice of similarity and merge functions are described in Table 1. Refer to Section 3.6 for the actual definitions of the similarity functions.

In general, $\mathsf{sim}_{\text{mult}}$ is the most common similarity function, since it maps a higher feature value to a more important vulnerability, by multiplying with some factor. It is a good fit for features ranging from less to more serious.

Considering a few special cases, such as access vector, the $\mathsf{sim}_{\text{dist}}$ distance similarity function is used instead, since this instead measures how close the feature value is to the user's preference. In this way, the user can select to rank e.g. local attacks higher than network-based attacks.

The Metasploit and publication date features have straightforward similarity functions based on their data type, while the CWE feature requires the use of the $\mathsf{sim}_{\text{cosine}}$ similarity to correctly handle the comparison between CWE vectors.

If we instead look at merge functions, a modified moving average $\mathsf{mer}_{\text{mma}}$ is used for most features, since it provides a simple way to converge towards to user's preference. For CWE, the special $\mathsf{mer}_{\text{add}}$ function needs to be used such that the vector of previously seen CWEs are merged with the newly rated CWE.

Finally, features with $\gamma_i = 0$ do not need merge functions, and are marked as N/A in Table 1.

5 Evaluation

In this section we present an initial evaluation of our recommender. The main purpose of the evaluation is to determine if the system fulfills its goals, which in our case is providing an assessment according to users' own preferences.

There are three common types of evaluation techniques for recommenders, namely user studies, online methods, and offline methods. In user studies, feedback is collected from users before, during, and after use of recommender. In online studies, information is collected from a running recommender, for example using A/B testing, so that results from two different groups with recommenders can be compared. Finally, in offline methods, a set of historical data is used to evaluate the recommender, without requiring ongoing interactions from users.

An online evaluation requires an already existing user base, which makes it difficult to use in our setting where we evaluate a new recommender. While offline evaluation methods are popular in recommender system evaluation [1], it requires the availability of historical data, which is domain specific. While data sets such as the Netflix Prize data set [14] is widely used, it cannot be used to evaluate a recommender system for vulnerabilities.

Because of the reasons above, we have decided to collect our own offline data set from users. This is similar to the user study approach described above, but the users do not actually use the recommender system, instead we ask them to manually provide their user profile, and rank a set of vulnerabilities. These results are then used as a data set to evaluate the recommender. Since the utility function applied to a set of vulnerabilities will induce a ranking of the vulnerabilities, we can use that ranking to determine how well our system succeeds in recommending vulnerabilities.

5.1 Evaluation Metrics

There are several different metrics used in recommender system evaluation. However, care must be taken to select metrics that are suitable to the type of recommender in question. Two common metrics are *precision* and *recall*, which

both measure the frequency with which a recommender makes relevant decisions. While common, these metrics are ill-suited for our recommender, since they consider other types of recommender system goals. In our recommender, the output is utility metrics for vulnerabilities, where it does not make sense to talk about precision and recall. Instead, we wish to measure the deviation between the recommender output and the actual rankings.

To do this, we have chosen predictive accuracy metrics and rank accuracy metrics, as these metrics are closer to the goal of recommender systems similar to ours [3, 17]. Predictive accuracy metrics measure how close the recommender system's predicted ratings are to the true user ratings [7]. Root Mean Square Error (RMSE) is probably the most popular metric used in evaluating accuracy of predictive ratings. The system generates predicted ratings $\hat{r}_{ui}$ for a test set L of user-item pairs (u, i) for which the true ratings r_{ui} are known.

The other type of metric, rank accuracy metrics, measure the ability of a recommender system to produce an ordering of items that matches how the user would prefer to have them ordered. To be able to evaluate based on rank accuracy, it is necessary to obtain reference ranking. We used the Yao's Normalized Distance-based Performance Measure (NDPM) [21] as rank accuracy metric, which calculates the difference between the order of items in preferred user order, and the system's recommendation order. The RMSE and NDPM can be calculated as follows [7]:

$$RMSE = \sqrt{\sum_{(u,i)\in L} (\hat{r}_{ui} - r_{ui})^2 / |L|} \qquad NDPM = \frac{C^- + 0.5C^{u0}}{C^u}$$

where C^- is the number of contradictory preference relations between the system ranking and the user ranking, C^{u0} is the number of compatible preference relations, and C^u is the total number of preferred relationships in the user's ranking. See [7] for details. The NDPM value varies between 0 and 1, where 0 means that the orderings are identical, and 1 means the ordering is reversed.

5.2 Experiment Results

In this section we want to evaluate the performance of the proposed recommender. In order to do this we selected a subset of CVEs, and then compared the recommendations made by the system, the manual ranking done by users, and the CVSS2 environmental scores.

For the evaluation, 8 users have been asked to participate. The users are working in the industry, for five different companies, and are people with high security awareness. These people are potential users of such a recommender. Each user started by selecting their own user profile, with preferences described in Section 4.2.

Then, 30 sample CVEs were selected, the CVEs were from different products, years, described different vulnerabilities, and were presented in a random order. The users were asked to rank these CVEs on a scale from 0 to 10, where a higher

value indicated higher interest to the user. The users were asked to only consider properties of the CVE itself, rather than the product it affected. To avoid bias from the CVSS base score, this score, as well as the impact and exploitability subscores, were hidden from the user during the evaluation. The users could however see other information in the CVE to make an informed decision.

After collecting the data, we proceed with the actual evaluation. The CVEs were divided into training and test sets using k-fold cross-validation, using $k = 5$. We performed an evaluation where both the user profile and the training set were used to train the recommender, before generating recommendations. As a comparison, we also compared the results to using the CVSS2 environmental score, with explicit user profiles mapped to impact subscore modifiers. For both cases, the reference ranking was the manual ranking performed by the users.

The RMSE and NDPM values were then calculated between the reference ranking and the recommender output, and between the reference ranking and the CVSS2 environmental score. The metrics can be seen in Table 2. We see that the RMSE values of the recommender system are lower compared to the CVSS environmental score. This indicates that the recommender has higher predictive rating accuracy for all users in comparison to just using the environmental score. The results also indicate higher rank accuracy in comparison to the environmental score based on the NDPM metric, for the majority of test users.

Table 2. RMSE and NDPM of recommender system and CVSS environmental score, relative the reference ranking, for different users

	RMSE		NDPM	
	Recommender	Environmental	Recommender	Environmental
User 1	0.179	0.222	0.303	0.287
User 2	0.247	0.340	0.195	0.271
User 3	0.200	0.256	0.207	0.333
User 4	0.153	0.296	0.179	0.276
User 5	0.168	0.286	0.294	0.283
User 6	0.138	0.234	0.175	0.228
User 7	0.115	0.224	0.147	0.251
User 8	0.198	0.267	0.349	0.340

6 Related Work

In [4], a vulnerability management system called VULCON was proposed. VULCON's objective is to reduce time-to-vulnerability remediation (TVR) and total vulnerability exposure (TVE) within an organization. VULCON takes inputs such as vulnerability scan data, target TVR requirements, and personnel resources. It then utilizes severity, persistence, and age of vulnerabilities to prioritize vulnerabilities. Compared to our paper, VULCON uses these three features,

while our recommender can utilize many vulnerability features. Furthermore, VULCON does not include any learning based on user history similar to ours.

Another recommender system has been suggested in [6]. Among others, the authors' describe a system which can speed up response to events such as cyber attacks. They use features such as the time since the vulnerability's discovery, severity of the exploit, existence of a patch, difficulty of deploying the patch, and impact of the patch on users. Compared to our paper, the authors does not at all discuss the construction of such a system. Their goal is also different, since their recommender should suggest an appropriate action on how to handle the vulnerability.

In [9], the authors present a method where they use textual description of vulnerabilities to construct a graph of related vulnerabilities. The authors' goal of producing a vulnerability ranking is similar to ours, but they do not discuss user-personalized rankings. The used features are also different: their recommender looks only at keywords from textual description, while we currently look at many other vulnerability features.

Previous work has also looked at designing different vulnerability metrics, as opposed to using the CVSS score. In [10] the authors proposed VRSS, a system to rate and score vulnerabilities, using a combination of qualitative and quantitative methods, resulting in scores closer distributed to the normal distribution. WIVSS [19] is a system with similar goals, where the authors propose a scoring system with the goal of more diverse scores and better accuracy. However, neither of these two vulnerability metrics consider individual user preferences as done in this paper.

7 Conclusions and Future Work

We have defined, implemented and evaluated a recommender system providing severity assessments of vulnerabilities. The recommender system is specialized for vulnerabilities, and is designed to be useful specifically for the context of vulnerability assessment. Recommendations are generated by considering both users' explicit preferences, and by considering their previous interactions with the recommender. The system can be used with a variety of different inputs, and can easily be extended with new features if desired.

The evaluation shows that the system gives better recommendations compared to just using the CVSS environmental score. To be able to tune the parameters for optimized performance, data from more users is needed. However, the results from our evaluation with real users suggests that it is possible to improve the assessment using a recommender system approach. Other possible future work includes consider negative feedback in the learning phase, which may further improve the results when learning is enabled.

Acknowledgements This work was partially supported by the Swedish Foundation for Strategic Research, grant RIT17-0035, and partially supported by

the Wallenberg Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg foundation.

References

1. Aggarwal, C.C.: Recommender Systems. Springer (2016)
2. Chen, L., Sycara, K.: Webmate: A personal agent for browsing and searching. In: Proceedings of the Second International Conference on Autonomous Agents. pp. 132–139. AGENTS '98, ACM (1998)
3. Degemmis, M., Lops, P., Semeraro, G.: A content-collaborative recommender that exploits wordnet-based user profiles for neighborhood formation. User Modeling and User-Adapted Interaction **17**(3) (2007)
4. Farris, K.A., Shah, A., Cybenko, G., Ganesan, R., Jajodia, S.: Vulcon: A system for vulnerability prioritization, mitigation, and management. ACM Transactions on Privacy and Security (TOPS) **21**(4) (2018)
5. First: Common vulnerability scoring system v3.0: Specification document, `https://www.first.org/cvss/specification-document`
6. Gadepally, V.N., et al.: Recommender systems for the department of defense and the intelligence community. MIT Lincoln Laboratory (2016)
7. Gunawardana, A., Shani, G.: Evaluating recommender systems. Recommender systems handbook, Springer, Boston, MA (2015)
8. Höst, M., et al.: Industrial practices in security vulnerability management for iot systems an interview study. In: Proceedings of the 2018 International Conference on Software Engineering Research & Practice. pp. 61–67 (2018)
9. Lee, Y., Shin, S.: Toward semantic assessment of vulnerability severity: A text mining approach. In: 1st International Workshop on EntitY REtrieval (EYRE '18) (2018)
10. Liu, Q., Zhang, Y.: VRSS: A new system for rating and scoring vulnerabilities. Computer Communications **34**, 264–273 (2011)
11. Mell, P.M., et al.: A complete guide to the common vulnerability scoring system version 2.0 (2007), `https://www.nist.gov/publications/complete-guide-common-vulnerability-scoring-system-version-20`
12. Meteren, R.v., Someren, M.v.: Using content-based filtering for recommendation. In: Proceedings of ECML 2000 Workshop: Machine Learning in Information Age. pp. 47–56 (2000)
13. MITRE Corporation: Cve details. `https://www.cvedetails.com/`
14. Netflix: Netflix prize. `https://www.netflixprize.com/`, (visited on: 2019-02-07)
15. NIST: National vulnerability database. `https://nvd.nist.gov/`
16. Rapid7: Vulnerability and exploit database. `https://www.rapid7.com/db`
17. Rosaci, D., Sarn, G., Garruzzo, S.: Muaddib: A distributed recommender system supporting device adaptivity. ACM Transactions on Information Systems (TOIS) **27**(4) (2009)
18. Smyth, B.: Case-Based Recommendation, pp. 342–376. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
19. Spanos, G., Sioziou, A., Angelis, L.: WIVSS: A new methodology for scoring information systems vulnerabilities. In: Proceedings of the 17th Panhellenic Conference on Informatics. pp. 83–90. PCI '13, ACM, New York, NY, USA (2013)

20. Synopsis Center for Open Source Research & Innovation: 2018 Black Duck by Synopsys Open Source Security and Risk Analysis (2018), https://www.blackducksoftware.com/open-source-security-risk-analysis-2018, Last accessed: 2018-11-08
21. Yao, Y.Y.: Measuring retrieval effectiveness based on user preference of documents. Journal of the American Society for Information Science **46**(2) (1995)