Recommending Security-related Documents

2.1 Information Handled by Security Teams

When running security teams, such as SOCs or CSIRTs, you inevitably have to deal with reams of information on a daily basis. The term "information" is a single word, but the concept is broad—the nature of information and how it is handled varies. In many organizations, personnel responsible for security collect, compile, and create the information needed to fulfil their respective roles.

From a systems development perspective, this information falls into two broad categories: structured information that is amenable to automated processing, and unstructured information.

Examples of structured information that can be used by security teams include IP address blacklists, TCP/IP port databases, and SCAPs formulated to automate security operations. Semantic, structured information is amenable to automated processing and is thus widely used in systems that support security.

Unstructured information, meanwhile, is also necessary in security operations but does not lend itself well to automated processing. It includes, for example, documents written in natural languages and images. Even in support systems, it has been difficult to do anything with this sort of information other than presenting it in document form as part of reference information. Unstructured documents and images are often simply accumulated within reports and the like created at the time the information was relevant, meaning that any subsequent use of that information comes down to manual effort.

How then should unstructured information be pulled up when necessary and referenced when relevant during, for example, the performance of security tasks?

2.2 Dealing with Unstructured Information

A number of technologies aim to solve this sort of problem. The first that comes to mind is full-text search systems. Anyone can appreciate the convenience to be had in collecting and storing documents that have been created in a full-text search system so that users can find information via keyword searches when required.

Another type of system is one that presents information of potential importance to the user without the user having to search for anything. These systems include those that provide product recommendations on shopping sites and list related articles on news sites. These types of systems are called recommender systems.

When looking to use these sorts of convenient, proven technologies with natural language documents that you have on hand, implementing all-text search systems was fairly easy, but the task of implementing systems that use collective knowledge, such as recommender systems, has posed difficulties. Much of the information necessary for security operations within companies is confidential, with its use being restricted to certain authorized personnel within the organization, meaning that there are few users to begin with and not enough data can be amassed to make use of collective knowledge.

With this background in mind, here we test out an approach to deciding on recommendations based solely on the information in unstructured documents and without using the user's action history. An advantage of the approach we took and describe in this article is that user actions and documents can be handled without being externally exposed. Let's look at the task of recommending documents that are related to one that the user has selected.

2.3 Natural Language Processing and Topic Models

What sort of technology is used to deal with unstructured documents written in natural languages? This sort of technology is called natural language processing (NLP), a field that has been studied for many years and comprises various component technologies. One of those is topic modelling, which uses machine learning to analyze textual data (Figure 1).

2.3.1 Topic Models

Documents come in various types. Even in the limited context of what we read every day, technical documents and news articles can be thought of as different types of documents. News articles also come in a number of types, such as those reporting on world affairs and those reporting on sports results. The type and frequency of language used may also vary according to the type of document. Moreover, any single document does not necessarily belong to only one type. A document about internet-based attacks sparked by international conflict, for example, would belong to both world affairs and information security.

In the field of topic models, the types to which a document belongs are referred to as topics, and documents are assumed to be generated in the following manner.

First, the document's topic mixture (the degree to which a document's topics are mixed) is determined according to some probability distribution. The document is then filled with words from each topic using the probability of the word's occurrence within that topic until the final document has been generated. If these topic and word probability distributions are calculated from actual textual data, they can be used to investigate what topics are currently being focused on and to categorize documents based on topics.

Methods based on this conceptual approach are collectively called topic models. Latent Dirichlet Allocation (LDA) is the archetypal model, with a range of variations having also been created and studied.

We can expect the topic distributions of closely related documents to be close to one another, so we will use this observation and take the approach of finding documents that have topic distributions close to that of the document the user has selected and displaying those as the recommendations. Here, we use LDA to compute the topic distributions.

2.3.2 Preprocessing with Natural Language Processing Techniques

You cannot simply pass a document list into the LDA algorithm. It also needs the frequencies of the words that appear in the text. In other words, documents require some preprocessing to make them suitable for the algorithm. At the same time, we also pare information considered unnecessary with the aim of both enhancing accuracy when generating the LDA model and to reduce the volume of data required.

The following preprocessing steps are needed.

1. Extract the body text from document data
   This entails removing parts of the document data that are not part of the main body text.
2. Split the text into words (tokenization)
   With English text, this can largely be accomplished by splitting text strings on spaces and line breaks. But a number of points also need to be addressed, such as line-breaking hyphens in text with line-length limitations. If you want to treat inflectional forms of a word as the same word[*1], NLP techniques called stemming

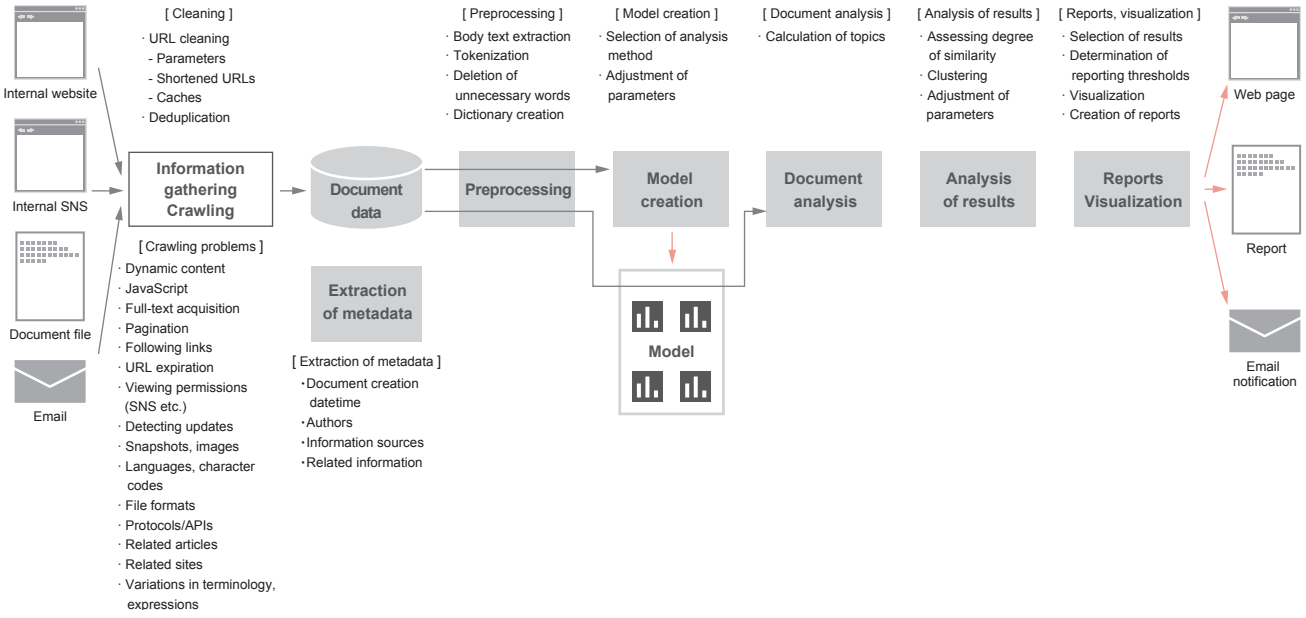

Figure 1: Overview of Natural Language Processing Using Topic Models

*1 E.g., word-words and write-wrote-written

and lemmatization are available. Japanese text does not contain any clear word barriers, so the tokenization process involves using morphological analysis.

3. Delete unnecessary words

   Words such as “a”, “an”, and “the”, for example, appear frequently in English text but have little to do with the overall meaning. Having the dataset filled with such words, which do not really seem relevant to the objective, is unlikely to improve the accuracy of analysis, so the usual procedure is to delete these words during preprocessing.

   One approach is to use a predefined list of stop words to delete. Other approaches include deleting words that appear frequently in the input text, and also deleting sparse words that appear only infrequently. The task of deciding which words it will be effective to delete is one of trial and error based on your objective and past examples.

4. Create a word frequency table for each document

   This involves counting the number of times words appear in the document. The word frequency table produced is called a bag of words. A separate frequency table is created for each individual document. The list of frequency tables is what you actually pass into the LDA algorithm.

As you will notice from this description, the bag of words does not reflect the order in which words appear in the documents, which means that LDA does not take word order or context into account.

CVE-2018-5383

Bluetooth firmware or operating system software drivers in macOS versions before 10.13, High Sierra and iOS versions before 11.4, and Android versions before the 2018-06-05 patch may not sufficiently validate elliptic curve parameters used to generate public keys during a Diffie-Hellman key exchange, which may allow a remote attacker to obtain the encryption key used by the device.

Figure 2: Example of a text file named by CVE ID and containing the vulnerability summary

2.4 Prototyping

We created prototypes to validate these technologies. Here, we use vulnerability summaries from the CVE[*2] database of vulnerabilities published by MITRE[*3], a US-based not-for-profit organization. Many natural language processing technologies for English text are available as part of libraries, so much of the process can be accomplished simply by using those libraries.

First, let’s prepare the source data. In our prototypes, we use the 7,692 CVE vulnerabilities released to date in 2018. We download the CVE data from the NVD Data Feeds[*4] page provided by the NIST in the United States. A beta release of the data in JSON format is available, but we use the XML feed on this occasion, as we have in the past. The dataset includes fields that are easily handled by computers—including CVE ID, publication datetime, datetime of last modification, and links to reference information—but we extract only the unstructured natural language description (found in the vuln:summary tag) for each vulnerability and save these in separate files named according to the CVE IDs (Figure 2). We now have the source text files ready.

2.4.1 Text Preprocessing

We use the Python libraries gensim[*5] and nltk[*6] to perform the processing steps required for creating the LDA model.

First, we perform the necessary preprocessing on the source documents to enable us to create the LDA model. Although not present in the CVE data we used in our prototypes, preprocessing steps commonly performed on English text lifted from websites include:

- Removing HTML tags and processing special HTML characters
- Joining words split by hyphens at line ends

Next, we tokenize, lemmatize, and remove stop words to create word data from the documents. Although we simply pass a string into the lemmatize function, internally the function performs a lot of complicated processing for us, including extracting

*2 CVE (https://cve.mitre.org/).
*3 MITRE (https://www.mitre.org/).
*4 NVD Data Feeds (https://nvd.nist.gov/vuln/data-feeds).
*5 gensim (https://radimrehurek.com/gensim/).
*6 nltk (http://www.nltk.org/).

only the nouns, verbs, adjectives, and adverbs and converting inflectional forms of each word to a common base form.

We perform the above steps on each of the documents, and create a list containing the extracted word data.

```
def normalize(txt):
    # De-hyphenation of words across a line-break
    txt = re.sub(r'-\n', '', txt)
    # Concatenate lines
    txt = re.sub(r'\n', ' ', txt)
    # Tokenization and lemmatization
    tokens = [ re.sub(r'/[A-Z]+$', '', x.decode('utf-8'))
                    for x in gensim.utils.lemmatize(txt) ]
    # Remove stop-words
    stopwords = nltk.corpus.stopwords.words('english')
    tokens = [ token for token in tokens if token not in stopwords ]
    return tokens

docs = []
for path in files:
    with open(path, encoding='utf-8') as f:
        txt = ''.join(f.readlines())
    tokens = normalize(txt)
    docs.append(tokens)
```

The gensim.utils.lemmatize() function that we invoke in the normalize function here appends part-of-speech information to the end of words, so we use a regular expression to remove this.

```
[b'cve/VB',
 b'high/JJ',
 b'rate/NN',
 b'vlan/NN',
 ...
```

2.4.2 Creating the Dictionary and Bag of Words

The gensim library that we use here assigns an ID to each word. It assigns word IDs based on the list of word data that we created and then counts the words in each document. This data structure is called a dictionary.

We then use gensim to analyze and filter the content of the dictionary. Specifically, we removed:

- Words that appear in many documents
  (Example: Words that appear in 20% or more of the documents)
- Words that seldom appear
  (Example: Words that appear in only one document)

We experimented by changing the parameters and omitting this processing step altogether. We observed a noticeable decline in categorization accuracy with models that we created without removing words that appear in many documents. When it came to the removal of words that only seldom appear, we did not observe any noticeable effect to the extent that we experimented.

Using the filtered dictionary, we create a bag-of-words (BoW) vector for each document. These are vector representations of the number of times words in the dictionary appear in the document.

```
dic = gensim.corpora.Dictionary(docs)
dic.filter_extremes(no_above=0.2, no_below=1)
bow = [ dic.doc2bow(doc) for doc in docs ]
```

2.4.3 Creating the LDA Model

We create the LDA model from the dictionary and bag of words, and then save the model and the associated data that we have created in a file.

When creating an LDA model, you need to specify the number of topics. The appropriate number of topics to use apparently changes depending on the source documents as well as the volume of words that appear and the way they are distributed. Various approaches to deciding on the value exist. We experimented by changing which documents, and how many, we fed into the model. With our data, we found that we were often able to create models that did relatively well by using values in the range of 30–50, and we therefore use 50 here.

```
lda = gensim.models.ldamodel.LdaModel(bow, id2word=dic, num_topics=50)

lda.save(filename_model)
dic.save(filename_dic)
gensim.corpora.MmCorpus.serialize(filename_corpus, bow)
```

2.5 Analyzing Documents Using the Model

We now analyze the documents using this model. The results allow us to see what sort of topics each document contains.

```
results = []
for doc in docs:
    bow = lda.id2word.doc2bow(doc)
    doc_topics = lda.get_document_topics(bow)
    results.append(doc_topics)
```

2.5.1 Pulling Up Similar Documents

Using the results of this document-wise analysis of topics and calculating the cosine similarity between vectors allows us to select documents that have closely similar topic components from among the set of all documents. So we selected a number of CVE vulnerabilities that had been in

the news and such recently to verify whether we could actually pull up similar vulnerabilities.

For example, when we looked at CVE-2018-8373 (Figure 3), which was fixed by a monthly Microsoft patch in August, we obtained a whole list of similar CVE vulnerabilities that also had to do with a "Scripting Engine Memory Corruption Vulnerability" (CVE-2018-0955, CVE-2018-0996, CVE-2018-1001, CVE-2018-8267, etc.).

```
[('CVE-2018-0955', 1.0),
 ('CVE-2018-0988', 1.0),
 ('CVE-2018-1001', 1.0),
 ('CVE-2018-8267', 1.0),
 ('CVE-2018-8353', 1.0),
 ('CVE-2018-8371', 1.0),
 ('CVE-2018-8373', 1.0),
 ('CVE-2018-8389', 1.0),
 ('CVE-2018-0996', 0.9999999),
 ('CVE-2018-8242', 0.9999997),
 ('CVE-2018-0839', 0.9968462),
 ...
 ('CVE-2018-8385', 0.9579928),
 ...
 ('CVE-2018-8372', 0.8273082),
 ('CVE-2018-8355', 0.8272891),
 ...
 ('CVE-2018-8359', 0.7355896),
```

Upon reading the documents determined to be similar, we did indeed observe many results that were similar enough to say that the documents followed an almost standardized wording.

When we looked at how similar the related CVE vulnerabilities listed in the CVE-2018-8373 summary were, we observed cosine similarities in the range of 0.73–1.0. However, 178 CVE vulnerabilities fell within this range. Although we can indeed say that we are able to find similar documents, it appears we may need some fine-tuning if we are to use these similarity scores to present documents of interest to the user.

Next, let's look at CVE-2018-3620 (Figure 4), which relates to an exploit known as Foreshadow-NG, a side-channel attack on CPUs that use speculative execution. Related entries in the list of similar CVE vulnerabilities that we calculated included CVE-2018-3646 and CVE-2018-3615. However, the list also included vulnerabilities related to a PHP chatbot program and a web application framework with similar cosine similarity scores. Using the cosine similarity of topics as the only threshold metric may not produce very good results.

```
[('CVE-2018-3620', 0.9999924),
 ('CVE-2018-3646', 0.9803927),
 ('CVE-2018-3640', 0.88989854),
 ('CVE-2018-3693', 0.8448397),
 ('CVE-2018-3615', 0.8389072),
 ('CVE-2018-5954', 0.8318751),
 ('CVE-2018-1000181', 0.80068177),
 ...
```

As demonstrated, we were able to determine that it is possible, to an extent, to find CVE entries that are along the same lines as any particular CVE vulnerability of interest by using a topic model and cosine similarity scores to find CVE entries with similar summaries.

That said, we observed many cases in which the model also pulled up a bunch of documents that did not appear to be very similar at all. And in some cases, the model failed to pull up some of the documents we had hoped for.

Take CVE-2018-5390 (Figure 5), a DoS vulnerability in the Linux kernel's TCP implementation. The documents at the top of the list of those determined to be similar did not include any entries with relevant content. The source documents included, for instance, 105 CVE entries on Linux kernel vulnerabilities, but they did not have high similarity to CVE-2018-5390.

```
[('CVE-2018-5390', 0.99944544),
 ('CVE-2018-1237', 0.81856203),
 ('CVE-2018-1240', 0.8044424),
 ('CVE-2018-1217', 0.80138516),
 ...
```

Apart from where we set the cosine similarity threshold, these results can also be influenced by the parameters used

A remote code execution vulnerability exists in the way that the scripting engine handles objects in memory in Internet Explorer, aka "Scripting Engine Memory Corruption Vulnerability." This affects Internet Explorer 9, Internet Explorer 11, Internet Explorer 10. This CVE ID is unique from CVE-2018-8353, CVE-2018-8355, CVE-2018-8359, CVE-2018-8371, CVE-2018-8372, CVE-2018-8385, CVE-2018-8389, CVE-2018-8390.

Figure 3: Summary of CVE-2018-8373

Systems with microprocessors utilizing speculative execution and address translations may allow unauthorized disclosure of information residing in the L1 data cache to an attacker with local user access via a terminal page fault and a side-channel analysis.

Figure 4: Summary of CVE-2018-3620

Linux kernel versions 4.9+ can be forced to make very expensive calls to tcp_collapse_ofo_queue() and tcp_prune_ofo_queue() for every incoming packet which can lead to a denial of service.

Figure 5: Summary of CVE-2018-5390

at each stage in the model creation process. Experiments we performed beyond what we have described here showed that the results can vary substantially depending on where these parameters are set. This also gave us insight into the difficulty of dealing with topic models, inasmuch as it is difficult to adjust each of the parameters before examining the results of the model in full.

2.5.2 Improving Accuracy by Using Peripheral Information

Topic models are one effective way of quickly finding similar documents from within a collection of documents written in unstructured natural language. That said, if the output results are to be judged on the basis of whether the model is able to properly pick out documents of interest to the reader, then some fine-tuning suitable to the application at hand will be needed to improve accuracy. This is because which similar documents will be relevant to the user differs according to the circumstances.

For instance, if you want group similar articles from the daily news, then everything other than the most recent information is likely to simply get in the way. Yet if searching for documents relevant to dealing with a problem that occurs only rarely, then surely the reader would also like older information to be included in the document search results.

To deal with this diversity, some academic research on topic models is directed at adjusting the structure of models in the aim of improving accuracy for specific applications. This includes, for example, models that incorporate analysis of time-series variations in topics and models that incorporate analysis of author names.

For this exercise here targeting CVE and security-related documents, we also experimented with filtering using peripheral information. For internal memos, for example, we were able to improve the accuracy with which we found documents of relevance to the user by giving priority to documents created around the same time as the document in question. When looking at internal documents, we found that refining the results based on keywords appearing in project names and the document path was effective.

We also looked at whether refining the results in a situation-independent manner could improve accuracy. We had some good results when filtering the list based on the results of topic clustering using clustering algorithms available in the Python scikit-learn[*7] library, such as DBSCAN. With this method, however, tuning is crucial because the parameters set when clustering greatly influence the nature of the clusters, which means it may be difficult to use this approach consistently in a situation-independent manner.

As discussed here, we conducted a range of experiments focusing on NLP and topic models in an effort to explore effective ways of dealing with unstructured information. Although we were unable to produce satisfactory results with any one method alone, we did discover that, by combining multiple methods according to the objective, it may be possible to fine-tune the approach and get closer to the desired output. Based on these findings, we will continue to look at applications of these methods to situations involving the use of unstructured documents under certain conditions.

Tadaaki Nagao

Senior Engineer, Office of Emergency Response and Clearinghouse for Security Information, Advanced Security Division, IIJ
Mr. Nagao joined IIJ in April 1998. Having worked on security services development, SDN development, and other roles, he is now engaged in research on information security in general from a theoretical perspective.
He is a member of IIJ Group emergency response team IIJ-SECT.

Yasunari Momoi

Lead Engineer, Office of Emergency Response and Clearinghouse for Security Information, Advanced Security Division, IIJ
Mr. Momoi joined IIJ in January 1999. Having worked on research and development of a range of services, including security services and wireless IC tag systems, he is now engaged in research and development related to information security in general.
As a member of IIJ-SECT, he participates in the activities and running of groups such as Information Security Operation providers Group Japan (ISOG-J) and ICT-ISAC.