Cyber Relation Extraction from Unstructured Documents

Isaac Persing

Hosting Site: Oak Ridge National Laboratory

Mentor(s): Kelly Huffer and Robert Bridges

**Abstract.** In order to protect information systems from threats, cyber security analysts require information from such varied sources as system logs, malware databases, news sites, and OS and application vendor blogs. However, there is no standard structure to how the data in these sources is stored, nor is there a central location from which it can be obtained. The Stucco project aims to make the analyst’s job easier by collecting data from these disparate sources into one, consistently-structured knowledge graph. During my internship, I worked on Stucco’s subtask of automatically extracting cyber security information from unstructured natural language documents like blog posts and news articles. I helped my mentors to set up a system for collecting cyber security relevant natural language documents in real time, and designed a system for extracting relationships between cyber security entities (e.g. things like software products and vulnerability names) appearing in the documents. I heuristically extracted instances of the (Software Vendor produces Software Product) relationship from 129 documents, then trained an SVM to predict this relationship in a supervised manner using several novel features. This system yields a relative error reduction of 10% in f-score over a baseline system which utilizes only standard NLP features.



Internship Project

The Stucco project aims to help cyber security analysts work more efficiently by providing them with cyber security-relevant information from a wide variety of different types of sources. The data from all these sources will be available from one central knowledge base, making it more easily accessible than the sources from which the data is collected because 1) analysts will not have to track down relevant information from multiple sources in multiple locations, and 2) the knowledge base will store all its data in one standard format, thus making manual inspection of sources unnecessary.

One of the types of sources the Stucco project’s researchers would like to incorporate into the knowledge base includes natural language documents like cyber security blog posts and news articles. The reason for their inclusion is that such documents often discuss the latest cyber security threats long before any information about them appears in vulnerability databases cyber security analysts normally rely on such as the National Vulnerability Database (NVD) or Common Vulnerabilities and Exposures (CVE). Thus, keeping the knowledge base up-to-date requires information from these sources.

Unfortunately, Stucco researchers suffer from the same difficulty as the cyber security analyst when it comes to natural language documents. In order to populate the Stucco knowledge base with information from these documents, a human would have to read each document to manually discover the important facts it reports in order to enter them into the knowledge base. But is this necessarily true? Would it be possible to instead teach a computer to read these documents and automatically extract the information we want to enter into the knowledge base? In my internship project, I attempted to answer this question by assembling a system for extracting cyber security information from these documents using natural language processing (NLP) and machine learning techniques.

By my second week at Oak Ridge National Laboratory (ORNL), my mentors and I had decided on a general approach to this problem. Broadly, our plan went as follows. First, we would set up a stream from which we would collect a large number of unstructured cyber security documents from the internet. Then we would identify the cyber security-related entities (e.g. software products like “Firefox”, or vulnerability names like “the thunderstrike issue”) appearing in these documents. Next, we would heuristically extract some of the relationships between these entities (e.g. relationships like that between a software product and piece of malware it is vulnerable to). Finally, we would train a learner on these heuristically-labeled relationship examples in a supervised manner using some machine learning algorithm. The model learned by the algorithm could in the future be leveraged to identify relationship instances from newly-collected documents to add to Stucco’s knowledge base.

Collecting Unstructured Cyber Security Relevant Documents

The first step toward training a machine learning model for identifying relationships is collecting the data on which it will be trained. Fortunately for my project, a system for collecting cyber security relevant documents had already been created prior to my arrival at ORNL. I merely needed to configure it to collect documents from appropriate sites.

Here too, I was helped by work that had been done by previous Stucco project researchers. Over the past four years, several researchers contributed to a list of cyber security relevant websites, many of which contained unstructured natural language text of the type we wanted to extract information from. With help from my mentor Kelly Huffer, I selected 42 of the websites from this list and wrote a configuration file instructing the aforementioned collection system to continuously extract new documents from them whenever they appear. So for example, since krebsonsecurity.com was one of the websites listed in my configuration file, whenever a new blog post appears on this site, our collector will extract its text.

Unfortunately for my project, the existing collector system had not been thoroughly debugged before my arrival, so I continually ran into errors while trying to collect documents. Normally when this happened, I would do a little investigation to document under what circumstances the error occurred, then send the description of the problem to my mentor who would fix it. So my role in the debugging process was mostly that of a tester.

There was, however, one of these problems that I managed to solve on my own. For some reason, the collector would not collect documents when given some versions of a configuration file. I was unable to discover exactly what circumstances caused this, aside from the facts that the error was far more likely to happen when the collector was instructed to collect documents from a large number of sites, and that there was no one particular site in my list of 42 that was responsible for the error. So I worked around the error by writing a shell script that iteratively restarted the collector, each time directing it to collect documents from a different site.

Identifying Cyber Security Relevant Entities

The second major step in the process of identifying relationships from unstructured documents to add to the Stucco knowledge base is identifying the entities in those documents that participate in the desired relationships. So for example, if the sentence in Figure 1 appeared in one of our documents, our system would be expected to identify that Microsoft is a software vendor and that Windows is a software product because the list of entity types we are looking for (file name, function name, software product, software vendor, software version, vulnerability CVE ID, vulnerability description, vulnerability MS ID, and vulnerability name) includes those two entity types.

|  |
| --- |
| *Microsoft released 11 security updates to fix vulnerabilities more than 40 flaws in Windows and related software.* |

Figure 1. An example sentence containing cyber security relevant entities.

I will not dwell on this step in the process because this is another problem that had already been solved before my arrival at ORNL, as described in [1]. So I had access to a pre-trained learner for identifying all these entities. My role in this step was merely as a user.

Heuristic Cyber Relation Extraction

The next step in the process of extracting relationships for Stucco’s knowledge base is to produce a training set for the machine learner we plan to produce. To do this, I performed heuristic cyber relation extraction on a large set of the previously-collected documents. Cyber relation extraction is the task of identifying the relationships that occur between the cyber entities appearing in a natural language document. The 24 pre-defined relationships we were looking for in these documents include relations such as those between a software product and the type of attack it is vulnerable to, or between the product and the name of its function where a vulnerability occurs. Figure 1 illustrates how the *software product is produced by software vendor* relationship may be expressed in our documents, and Table 1 shows some of the 24 relationships a cyber entity relation extraction system can identify and label.

**Table 1.** Sample Relationship Types

|  |  |  |
| --- | --- | --- |
| Cyber Entity 1 | Relationship | Cyber Entity 2 |
| Software Product | Is produced by | Software Vendor |
| Vulnerability Name | Is a vulnerability of | Software Product |
| Software Version | Has a vulnerability in | Function Name |
| Vulnerability Name | Occurs in | Function Name |

Heuristic cyber entity relation extraction, then, is the process of performing this task in a rule-based manner. More specifically, I heuristically identified *positive instances* (as exemplified in Figure 1) and *negative instances* (as exemplified in Figure 2) of each relationship from the documents, as this is the data that a traditional, supervised machine learning system needs in order to learn to make its own predictions on new documents.

|  |
| --- |
| *The latest, patched Flash version 22.0.0.209 for Windows and Mac systems is still available.* |

Figure 2. A negative example of the software product (Flash) is produced by software vendor (Mac) relationship.

To produce heuristically-labeled instances of a particular relationship, I first built a list of known relationships of all 24 types with the help of the National Vulnerability Database. This subtask was costly in itself, as I had to write custom rules for extracting each relationship type from the database dump. As an example of one of the more complicated rules, one of the entity types we are interested in is a vulnerability’s name. However, vulnerabilities are not explicitly assigned names in the NVD. They are only assigned an ID code. But I found that the vulnerability summaries in the database sometimes ended in text like “aka print spooler service impersonation vulnerability”. Whenever a vulnerability record’s summary ended in “aka”, I extracted the text after it as the vulnerability’s name, pairing it with all other entities mentioned in other fields of the vulnerability’s record if their types are paired with vulnerability description in any of our 24 relationships.

Recall that in the previous section we identified all the cyber entities appearing in our collected documents. Now for each relationship type, we select from our documents all the sentences that mention both types of entities that participate in the relationship. So for the “is produced by” relationship, we would select both the sentences in Figures 1 and 2 because both mention both a software product and a software vendor. Then, for each sentence, we use the list of relationships built in the previous paragraph to decide whether the sentence illustrates a positive or a negative instance. If the list tells us that the entities are related, the instance is positive. My mentor Dr. Robert Bridges helped me figure out how to determine if instances should be labeled negative. If both entities appear in the list in this relationship type, but the list does not tell us that they are related to each other, it is a negative instance. Otherwise, we treat the instance as having an unknown label, and we do not present it to our learner in the next step.

I use Freebase [2] to replace each entity in the documents and lists with a canonical form in order to make this process robust to the use of different aliases. So, for example, we can recognize that both “Internet Explorer” and “IE” are produced by Microsoft.

Automatic Cyber Entity Relation Extraction

While I have described how I heuristically identified positive and negative instances from cyber security relevant documents, I have not mentioned how I represented these instances to a learner. In machine learning, it is typical to extract *features,* instance attributes having some numerical value, from each instance which are used to map the instance to a point in a large dimensional vector space. The machine learning algorithm is then responsible for learning which points in the space could be mapped to positive instances, and which points could be mapped to negative instances. I used several natural language processing techniques to extract features from the previously-identified instances.

Word n-grams are the simplest type of feature I used to describe the instances. To illustrate how they work, consider all the text appearing in an instance’s sentence before the first entity. For each word in this segment of text, I create a feature indicating its presence or absence. So the “is produced by” instance from Figure 1 would have the feature associated with the word “patched”’s value set to 0, while the instance from Figure 2 would have this feature’s value set to 1. I produce features for each sequence of 1, 2, or 3 words appearing in this context window before the first entity, and additionally produce features for the word sequences appearing between the two entities and features for the word sequences appearing after the last entity.

A second, less simple but commonly used feature I used to encode instances is parse tree paths. To illustrate it, consider the syntactic parse tree for the sentence “Adobe issued a critical update for its Flash Player software.” shown in figure 3. If the entities we are interested in in this sentence are “Adobe” (software vendor) and “Flash Player” (software product), we would extract the feature NNP-NP-S-VP-S-VP-PP-NP-NNP because this is the path in the parse tree between “Adobe”, and “Player”, which is the last word in the second entity.

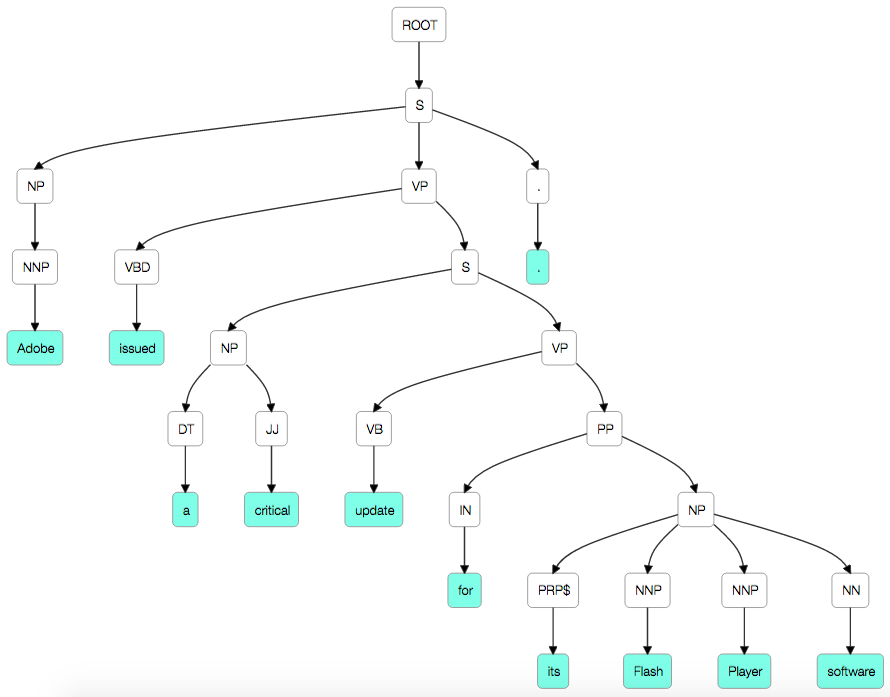


Figure 3: Syntactic parse tree example

The third and fourth feature types I encode are comparatively simple. The third feature type I encode counts the numbers of other entities of each type appearing in the same sentence before the first entity, between the entities, and after the second entity. The fourth feature type counts the number of words occurring before the first entity, between the entities, and after the second entity.

The fifth feature type I encode involves dependency parses. Particularly, I encode the dependency path between entities as a feature in three ways using the sentence’s dependency parse, illustrated in figure 4). One way simply lists the ordered edges from the first entity to the second (<-nsubj->ccomp->nmod->compound). The second way lists the nodes on this path (issued-update-software). Finally, the third way lists both, alternating between nodes and edges (<-nsubj-issued->ccomp-update->nmod-software->compound).

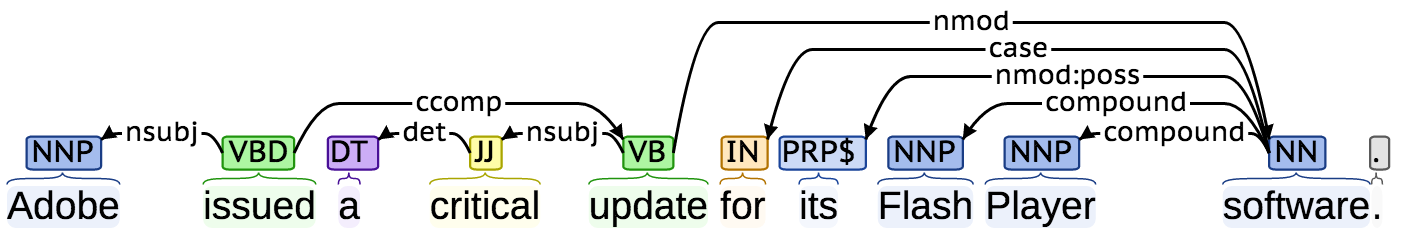


Figure 4: Dependency parse example

Finally, the most sophisticated feature I use involves *context embeddings.*  In order to build features of this type, I first needed to train a word2vec [3] model. A word2vec model represents words as vectors so that semantically similar words (e.g. dog and cat) appear near each other in the vector space, while semantically dissimilar words (e.g. dog and physics) do not. I trained a word2vec model on Wikipedia articles. Then I collected the words appearing before the first entity, averaging their associated word2vec vectors. The resulting vector’s values are used as features for the instance. I do the same for the context between the entities and the context after the entities.

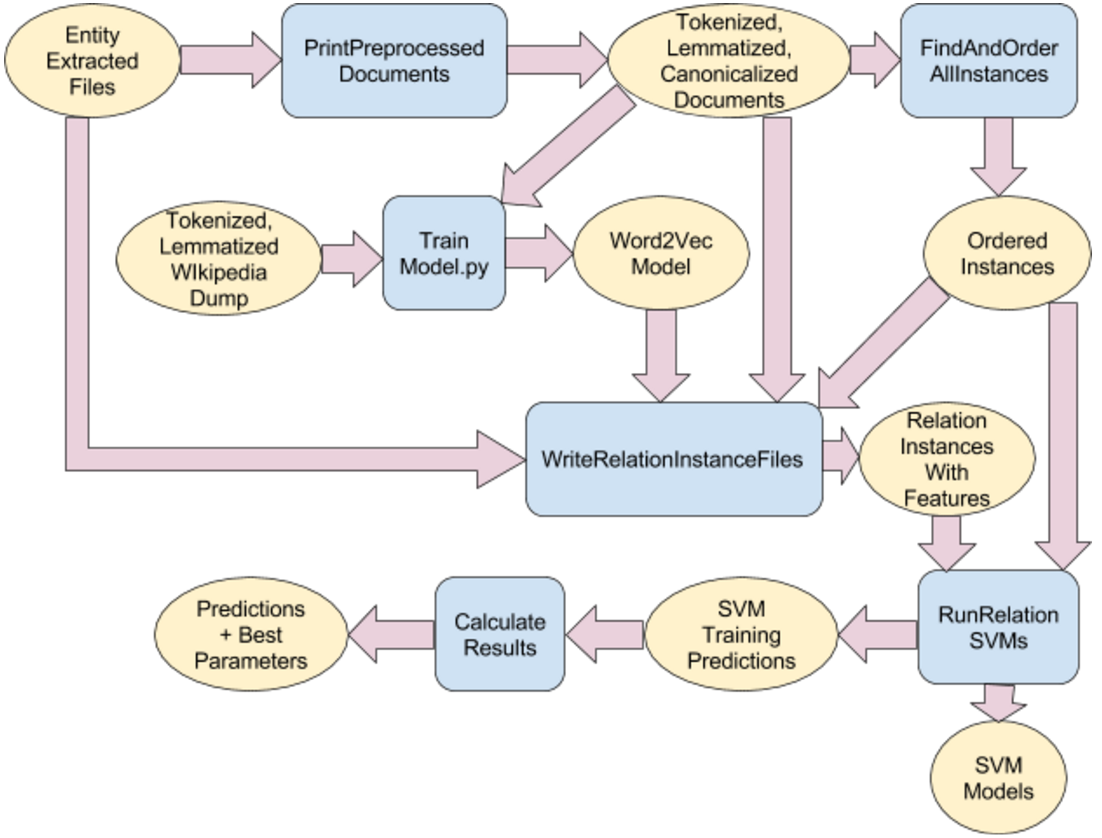
Evaluating Performance

It would have been sufficient to merely implement the relation extraction system I described since our goal is just to be able to predict whether a pair of entities appearing in text are in a relationship or not. However, it is always desirable to get an idea of how well a system works. For this reason, I performed 5-fold cross-validation on our set of 256 heuristically labeled “is produced by” instances, using the Support Vector Machine (SVM) machine learning algorithm. I found that when using all the features described in the previous section, my system obtained an f-score of .491. A baseline system that used only word n-gram and syntactic parse tree path features obtained an f-score of only .436. Thus my additional features were responsible for a 10% relative error reduction in f-score.

While my system’s performance may not be particularly inspiring, I would like to note that the fact that the training set was so small is probably largely responsible for the low scores. 256 instances is the smallest training set I have ever worked with in any NLP task I have ever tried to address. Given that there are so many different ways a writer can express one of our relationships, for this problem it is particularly important to have a large training set. Once we can automatically build a much larger training set, we should get better performance.

The Whole System

From a scientific standpoint, the features I briefly described above are the important contribution of my work to this project. However, a big part of my work at ORNL was to implement the programs that do all of this work (collecting instances, heuristically labeling them, encoding them as feature vectors, and machine learning a model for making relationship predictions) automatically. Figure 5 illustrates how data flows through one piece of the system I programmed. It is not readily understandable without the original accompanying documentation (available at <https://github.com/stucco/relation-bootstrap> ), but you should be able to tell from looking at it that the project required writing a lot of different programs (blue boxes), and ensuring they could communicate with each other through shared files (yellow ovals).



**Figure 5**: Illustration of data flow from entity-extracted documents to relationship predictions.

Impact of Internship on My Career

In addition to the cyber security tasks described more thoroughly in the first section, this internship introduced me to a lot of smaller tasks and tools that I would not have otherwise encountered until entering the workforce. I parsed xml and json files, which, though fairly common, I do not see in my work at UT Dallas. I learned how to work directly with compressed files since storage space was an issue in this project. I was introduced to Git and Maven, two very powerful tools for working on shared projects which I was only vaguely aware of before because I do a large majority of my research alone. For that reason, I never needed to make sure a shared project worked on any machine it is located on, so I learned tricks for ensuring programs from shared projects always know where to find files they need to access.

Some more NLP-specific things I got to work with were word2vec and Gensim [4]. Word2vec models, which I described in a previous section, are useful for a variety of NLP tasks, and working on this project has made me more likely to use them on other tasks in the future. Gensim is just the tool I used to create word2vec model, though in the process of learning how to use it, I discovered that it is a pretty useful tool for several other NLP tasks involving vector space models, so I am likely to use it in my work in the future as well.

I made Figure 5 to illustrate how data flows through my relation extraction pipeline for any of its future users. I had never previously made such an illustration, but during my internship I learned how to use Google Docs to put illustrations like this together easily. It seems likely that I will one day need to make another diagram, so this should be useful to me in the future. I also got to do some Python programming, which I had only done a little of before, and this is definitely something I will need again someday soon. During this project, I also worked with a couple virtual machines, which I had never done at all before. I am not sure how generally useful this will be because I do not have a feeling for how commonly they’re used, but it was a good experience to have, regardless.

A couple of the events I attended while at Oak Ridge included a presentation on effective poster design and a panel on early career development. Even though I did not create a poster during my internship, I have had to create posters for several of my publications before, though I was never exactly taught how to make one. My future posters should be more visually appealing since I attended this event.

The early career professionals panel taught me what it is like to be a new scientist at laboratories like Oak Ridge and how to get there. Though this was not its purpose, I think it made me less rather than more likely to pursue this type of career. I do not think all the travel these early career professionals described as part of their jobs would be to my taste. The panel made me reaffirm my commitment to pursuing a job closer to home. Since there are many defense contractors in my home town, in addition to all the things I described learning above, the mere fact that I had an internship through the Department of Homeland Security at Oak Ridge Laboratories may help me to this end.

References

[1] R. Bridges, C. Jones, M. Iannacone, K. Testa, and J. Goodall. Automatic labeling for entity extraction in cyber security. In Third Annual ASE Cyber Security Conference. ASE, 2014.

[2] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In SIGMOD, 2008.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient Estimation of Word Representations in Vector Space. In Proceedings of Workshop at ICLR, 2013.

[4] R. Řehůřek and P. Sojka (2010). Software framework for topic modelling with large corpora. In Proceedings of LREC Workshop on New Challenges for NLP Frameworks, 2010.