Automated Extraction of Vulnerability Information for Home Computer Security*

Sachini Weerawardhana, Subhojeet Mukherjee, Indrajit Ray, and Adele Howe

Computer Science Department, Colorado State University,
Fort Collins, CO 80523, USA
{sachini,mukherje,indrajit,howe}@cs.colostate.edu

Abstract. Online vulnerability databases provide a wealth of information pertaining to vulnerabilities that are present in computer application software, operating systems, and firmware. Extracting useful information from these databases that can subsequently be utilized by applications such as vulnerability scanners and security monitoring tools can be a challenging task. This paper presents two approaches to information extraction from online vulnerability databases: a machine learning based solution and a solution that exploits linguistic patterns elucidated by part-of-speech tagging. These two systems are evaluated to compare accuracy in recognizing security concepts in previously unseen vulnerability description texts. We discuss design considerations that should be taken into account in implementing information retrieval systems for security domain.

Keywords: security, vulnerability, information extraction, named entity recognition

1 Introduction

Hardening a home computer against malicious attacks requires identifying the different ways in which the system can be attacked, and then enacting security controls to prevent these attacks. This, in turn, requires deeper analysis to understand the contribution of different vulnerabilities towards an attack on the system, the role of a specific system configuration, the actions that an attacker must take to exploit the vulnerabilities, and, perhaps most importantly, the advertant / inadvertent contributions that home-user activities make in a successful exploit.

Researchers, including our group, have been investigating ways to capture these relationships and merge them into models of system security risk. Online vulnerability databases, such as the National Vulnerability Database (NVD - `http://nvd.nist.gov`), contain a wealth of information that is needed for creating these system risk models. Unfortunately, these vulnerability databases

* This material is based upon work supported by the National Science Foundation under Grant No. 0905232.

(VDs) have been created by humans primarily for the use of humans. A major problem is that VDs store vulnerability descriptions as natural language plaintexts. Important information, such as attacker actions and user actions, are seldom explicitly stated but rather remain implicit. A human expert is able to infer this from the description but this is challenging for an automated tool. Automating the extraction is further complicated by the fact that these descriptions vary significantly in how they describe the different pieces of information.

Roschke et al. [12] investigate the problem of extracting vulnerability information from semi-structured plaintexts with a goal of comparing and merging vulnerability information from multiple sources. However, this scheme requires developing customized extraction rule-sets for each individual databases, and is limited in its ability to extract information buried in natural language phrases. Urbanska et al. [15] also describe a similar approach for extracting information from VDs, based on designing customized extraction filters. Although this approach is able to extract crucial information such as attacker actions, it performs very poorly in extracting user action information. Moreover, the accuracy of the approach is heavily dependent on the quality of filters developed, which is a manual process.

A Named Entity Recognition (NER) system, employing a large corpus of hand annotated data extracted from different sources (vulnerability databases, blogs and news articles) is described in [5, 11]. A Conditional Random Fields (CRF) classifier [17] is trained on the corpus to identify portions of the text describing the concepts. Each portion is then processed to build associations between the concepts using an IDS Ontology specifically created for the security domain. A problem with this approach is that it is not able to extract information such as user or attacker actions. A bootstrapping algorithm called PACE is used for NER in computer security [10]. Its learning algorithm is pre-trained on a small set of known entity names (Exploit Effect, Software Name, Vulnerability Potential Effects, Vulnerability Category) and known patterns. A limitation of this approach is that the model needs to be trained on a large corpus (usually in the millions).

In this paper, we describe our efforts to design a tool to automatically extract severity information from natural language descriptions. Our work is designed to populate a model that we had proposed earlier [16], called the Personalized Attack Graph (PAG). Our approach is based on designing a NER that identifies key PAG parameters embedded in the text of a vulnerability description : software name, version, impact, attacker action, and user action. We experiment with two different NER approaches, one using machine-learning (ML) techniques, and the other exploiting lexical patterns in a parts-of-speech (POS) tagged text. The two approaches are then evaluated to compare their accuracy in recognizing security concepts embedded within newly encountered vulnerability descriptions.

Two labeled corpora are available for security vulnerability descriptions [1, 5]. We use Joshi et al.'s corpus [5] to validate the performance of one of our approaches that is similar to their approach. However, the PAG representation requires identification of security concepts not supported by Joshi et al.'s corpus,

most notably attacker and user actions. We, therefore, constructed our own corpus comprising of 210 randomly selected vulnerability descriptions taken from NVD as of April 2014. We used the BRAT web based annotation tool [7] to manually annotate the descriptions with the labels required for our PAG model. The annotation task was done by a computer science graduate student with substantial knowledge of computer security. We chose BRAT because it could be easily configured to allow for annotation of custom entity types. Our corpus consists of 8409 labeled tokens: 3821 default label tokens and 4588 security concept label tokens.

2 Our Approach to Extracting Computer Vulnerability Information

We implemented two independent NER solutions: a machine learning module and a part-of-speech, rule-based component. Machine learning is less brittle and may be able to accommodate examples never before seen. However, it requires training examples and a well selected set of features to support generalization. Rule systems can be efficient and well tailored to the domain; however, the accuracy drops quickly when new examples fail to match previous ones.

2.1 Machine Learning Module

The process of producing a machine-learning (ML) model for the task can be divided into two parts: feature selection and model training.
Feature Selection: We adopted the same features as in [5]: Taggy-sequences, N-grams, UsePrev, UseNext, Word-pairs, and Gazette. For the gazette, we compiled a list of software and operating systems consisting of 48709 entries using the product information repository available at www.secunia.com.
Model Training: We chose the Stanford Named Entity Recognizer (Stanford NER), that implements a CRF classifier [4]. It has the chosen features already built-in, and its CRF classifier has been widely adopted for custom NERs [13]. CRF takes into account contextual information by considering sequences of tokens, a property that can be specifically exploited by the structure of vulnerability descriptions.

2.2 Part-of-Speech Tagging

POS tagging allowed us to discover patterns in the grammatical structure of a vulnerability description and define a set of rules.
Identifying Software Names and Versions: POS tagging labels a word in text based on its role in the sentence (e.g., noun, verb, adjective, etc.) and the context in which it is used. We manually analyzed a sample of 30 vulnerability descriptions from NVD to identify persistent patterns of POS. We found that software names are typically tagged as `NNP` (proper nouns) and version numbers

are tagged as `CD` (cardinal numbers). This rule was applicable to 100% of the sample.

Software names are often followed by an `IN` tag (a preposition or subordinating conjunction), e.g., "Adobe Reader *before* 10.3". About half (45%) of the descriptions followed this rule. A variation to this rule happens when a software name is preceded by an `IN` tag and followed by another `IN` tag (e.g., "index.html in Mozilla Firefox 5.4 through 6.0"), which accounted for another 30% of the examples. For 17%, a software name is immediately followed by a `CD` tag (e.g., "Adobe Reader 9.3"). The remaining 6% were rare or erroneous structures (e.g., software name without a version number), which suggests that it would be difficult to achieve 100% accuracy.

Our POS tag processing algorithm is based on the observed patterns. We use 3-grams because they best match the length of common software names (e.g., "Adobe Acrobat Reader"). The Stanford coreNLP POS Tagger [14] provide the POS mappings. For each sequence of 3-grams, the algorithm checks for the specific patterns. If one of the patterns is found, then a gazette(see section 2.1) lookup is performed to identify the selected `NNP` tag as a software name. If none of the conditions are satisfied, a regular expression matching is performed. The 3-grams are then processed to identify any tokens tagged as `CD`s to find version numbers.

Identifying File Names, Modifiers and Vulnerability Type: File names in vulnerability descriptions typically denote a specific system artifact in which vulnerabilities are present. We try to identify modifiers, i.e., words that are used to indicate specific information about vulnerable software versions, as well as vulnerability types. File names are matched to a regular expression for `base_name.file_extension`:

`[[A-Z][a-z]{1, }]*[_-]?[[A-Z][a-z]{1,}0-9]*\\.[A-Z]?[a-z]{1,4}`

Modifiers typically follow version numbers (e.g. "version 4.0 through 5.1") and are tagged as `IN`. Each description is separated into POS Tag - Value pairs (e.g. {`before, IN`},{`2.0, CD`}). To find phrases with version numbers, the description is scanned to find words that have been tagged as `CD`. If a word tagged as `CD` is found in the description, POS tag of the word preceding a version number is checked. If this POS tag is of type `IN`, then the associated word is identified as a modifier. Identifying vulnerability type was done by searching for keywords: "vulnerability", "vulnerabilities" and "in". Then, the adjacent adjectives (POS tag `JJ`) are extracted.

Identifying Attacker Actions, Impacts, User Actions: To extract attacker, user actions and impact entities, we first partition a description into separate sentences and then discard information such as "*aka Bug id ...*","*related to ..vulnerability*" etc., which we know for sure can not be included as a part of the final result.

Identifying human actors is a two step process, which includes the use of the Stanford Parser [6], the Stanford Typed Dependency Representation [9], and WordNet Glosses [3], via the API RiWordNet(`http://rednoise.org/rita/reference/RiWordNet.html`). First, we identify actors as the nominal subject,

agent or the direct object of a verb. In case of a passive nominal subject dependency, we consider the passive agent as an indirect actor. We extract the WordNet Glosses [3] for these subjects, and if the glosses contain terms like "*human*","*person*" or"*someone*", we classify these as human actors.

However, at this point, we do not have enough evidence to identify the indirect actors as humans. Hence, we perform a search across the generated parsed dependency tree, pushing verbs into a stack, if they are directly related to both the passive nominal subjects and their governing verbs. As each verb is popped from the stack, it is verified whether its dependent is a non-human nominal subject or agent. If so, the concerned indirect actor is considered non-human and the iteration ends; else the verb is added to the stack and the iteration continues. If at the end of the search process, an indirect actor is found to have no direct or indirect relation with another non-human subject or agent, it is added to the list of human actors for further processing.

Each actor is attributed with a set of verbs and modifiers which are directly or indirectly related to this actor. If the actor is indirect, the set of verbs and modifiers for that actor can be enumerated as: $humanActor.VerbList \leftarrow \{governing\ verb,\ open\ clausal\ complements\ of\ governing\ verb\}$ and $humanActor.ModifierList \leftarrow \{adjective\ modifiers\ for\ direct\ object\ of\ governing\ verb\}$ respectively, else, as $humanActor.VerbList \leftarrow \{governing\ verb,\ reduced\ non-finite\ verbal\ modifiers\ of\ the\ dependent,\ open\ clausal\ complements\ of\ governing\ verb\}$ and $humanActor.ModifierList \leftarrow \{adjective\ modifiers\ for\ direct\ object\ of\ governing\ verb,\ adjective\ modifiers\ of\ the\ dependent\}$ respectively.

Since, in vulnerability summaries, the actor is not always referred to as "attacker", our next step involves identifying the actor as a "malicious attacker" or a "benign user"/"victim." In these cases, we analyze the sentiment value, obtained using SentiWordNet [2], of each of the verbs and modifiers attributed to the actor. This step assumes that the actors are malicious if the set of verbs or modifiers attributed to them contains at least one verb or modifier with a negative sentiment. We assign such actors as attackers, and others as victims.

To identify attacker actions, impact, and user actions, we consider each verb that has been attributed to each actor and identify the minimal verb phrases or sub-sentences they belong to. The term "minimal phrases/sub-sentence" refers to phrases or sub-sentences that do not contain any nested phrases or sub-sentences of the same type. Each phrase/sub-sentence is then considered individually and added to the *impacts*; any nested minimal verb and/or preposition phrase is extracted from it. The nested verb phrases are tested for starting verbs like "resulting" and "using" to denote them as *impact* or *attacker-action* respectively. The preposition phrases are first extended to the end of the verb-phrase/sub-sentence, and then verified for a starting preposition like "by", "through", "with", "via", etc. to be judged as *attacker-action*.

Finally, we perform a clean-up procedure. In this phase, for all actors, and for the three categories – user actions, impacts, and attacker actions – we verify whether any of these strings is a complete sub-string of another. If so, we

completely remove the smaller string from the larger string. However, we do not remove an *attacker action* from a different *attacker action* if both of them belong to the same actor. If any of these strings contain a minimal noun phrase enclosing the noun that identifies the actor, that section is also removed. Also, if the final strings start or end with stopwords and/or whitespaces/punctuations we remove them.

3 Evaluating the Extraction

A completely automatic extraction process requires a level of natural language understanding that is not currently feasible. Our evaluation focuses on what can currently be done. Thus, we ran two experiments to examine the following questions: What is the accuracy of the two approaches? How much can we reasonably automate? Are some concepts harder to automate than others? Are the two approaches complementary, favoring a hybrid approach as was done in [5]? The experiments were:

1. Validate that the performance of the re-implementation of the NER solution is similar to results reported by authors in [5].
2. Compare the performance of the two approaches with a focus on identifying trade-offs and possible complementarity.

We compute precision (Prec), recall (Rec) and F-measure (F1) [8] on a particular testing data set and for different entity labels to determine whether some labels are more difficult to automatically extract and whether the approaches differ in how accurately they extract each type.

3.1 Validate Implementation

The performance reported in [5] was computed on their own corpus (referred to as "Joshi corpus".) As the first part of validation, we computed the performance of our ML approach using the "Joshi corpus" and following their procedure (five-fold cross-validation) as closely as possible. Next, we trained our ML approach on our corpus and compared the performance to see whether the corpora differences led to significant differences in performance.

The most salient difference between the two corpora was in the labels extracted; because our labels were derived from the PAG model, our token set and "Joshi corpus"'s were not identical although had considerable overlap. Performance was calculated based only on the labels in the intersection: software, operating system, file name, NER-modifier, and consequence/impact.

Validation Experiment Setup: The Joshi corpus was partitioned into five equal sized sets: four sets for training and one set for testing. This corpus comprised four different sources – NVD, security blogs, Microsoft product specific vulnerabilities, and Adobe product specific vulnerabilities; the partitioning ensured that each source contributed uniformly to the set of descriptions in each

partition. Our corpus (which we call "NVD") included labels from 210 vulnerability descriptions extracted from NVD. Both corpora were trained using the CRF Classifier with 5-fold cross validation..

Validation Experiment Results: Table 1 shows the results reported in [5] ("Orig") compared to the results of our ML approach (mean and standard deviation over the five folds) when trained/tested on different corpora. Our solution on the Joshi corpus performs similarly to the reported results on Operating System and File (difference is less than the standard deviation); our precision was lacking on NER-Modifier and Impact.

For each metric (precision, recall and F1) we ran two-tailed t-tests to test for statistically significant differences in the accuracies. At the $\alpha < 0.05$ level, we found significant differences in eight out of 15; however, a Bonferroni adjustment would reduce the threshold to 0.003 leading to significant differences on only NER-Modifier and F1 for File. While we cannot use this analysis to confirm that there is no difference, it is likely that the differences were due to the disparate sizes of the corpora and to differences in the sources. Performance on NER-Modifier and Impact were improved on our corpus, while the others were worsened.

Table 1. Accuracy metrics and t-test results for validation. Orig is as reported in [5].

			Joshi Corpus		NVD Corpus		t-test
Label	Metric	Orig	Mean	SD	Mean	SD	$P <$
File	Prec	1.00	1.00	0.00	0.96	0.09	0.35
	Rec	1.00	1.00	0.00	0.35	0.22	0.36
	F1	1.00	1.00	0.00	0.58	0.14	0.002
Impact	Prec	0.71	0.45	0.08	0.58	0.08	0.20
	Rec	0.69	0.57	0.09	0.79	0.11	0.01
	F1	0.70	0.54	0.08	0.67	0.09	0.04
NER Modifier	Prec	0.79	0.48	0.15	0.94	0.02	0.002
	Rec	0.67	0.61	0.12	0.82	0.40	0.002
	F1	0.72	0.52	0.13	0.96	0.01	0.001
Operating System	Prec	0.95	0.91	0.04	0.54	0.40	0.25
	Rec	0.95	0.97	0.01	0.63	0.40	0.29
	F1	0.95	0.94	0.03	0.56	0.39	0.24
Software	Prec	0.86	0.65	0.05	0.53	0.05	0.008
	Rec	0.84	0.81	0.08	0.75	0.03	0.13
	F1	0.85	0.72	0.06	0.62	0.04	0.01

3.2 Compare Approaches

The second experiment compares the relative merits of our two approaches for the labels needed to represent our PAG model.

Comparison Experiment Setup: Essentially, the same procedure was followed for the ML approach as in the validation experiment. To assess the POS

approach, we followed the same procedure except that no training was required; the POS approach was tested on the test set for each of the five folds. The label set was expanded to encompass other labels in our corpus : attacker action (e.g., send a crafted image file), user action (e.g., opens a crafted image file), version and vulnerability (e.g., buffer overflow).

Comparison Results: Table 2 shows the results for each token type and approach. Component has been omitted because it was not implemented for POS; the reason was that there was no unique POS pattern to identify "components" in vulnerability descriptions. For ML, $Prec = .22$, $Rec = .29$ and $F1 = .25$. As shown by "Diff", POS is usually more accurate (in 22 out of 27 cases Diff is negative). ML has better precision for File and Operating System, better recall for Version and better F1 for Operating System and Version. This is somewhat surprising given that POS was constructed based on analyzing a small number of descriptions and encoding the observed patterns. On the other hand, it is using considerable knowledge about the language.

A two-tailed t-test on each of the metrics comparing the accuracy of the two approaches overall shows statistically significant differences in 15 out of 27 using $\alpha = 0.05$ and 6 out of 27 using a Bonferroni adjustment of 0.002. Although using POS is most often effective, the five cases where ML excels suggest that the combination may lead to slightly high accuracy. Overall the accuracy is good; max ranges from 0.27 for F1 for User to 0.99 for Precision for NER-Modifier.

Both approaches had difficulty in identifying user-action concepts. This is because vulnerability descriptions we analyzed do not explicitly mention user action in the description. For this reason, from the dataset we collected to train and test the learning model, we observed precision, recall and F1 measures of 0. However, false positives were generated in some of the scenarios for both machine learning and POS solutions. The reason for this, in the case of the POS tagging solution, is inaccurate sentiment values obtained from SentiNet 3.0. Since we rely on sentiments corresponding to verbs related to an actor in a sentence or its modifiers, if the sentiment of each word is either positive or neutral, even though the subject "user" in the sentence acts as the attacker, the action of that subject is classified as user action rather than attacker action.

4 Conclusion and Future Work

This work presents two approaches to automatically extracting information related to vulnerabilities that is buried in the natural language plaintexts in vulnerability databases – a machine learning approach and a POS tagging approach. Our experiments show that the POS approach is generally better (*Prec.* and *F1*) at identifying implicit entities in vulnerability descriptions such as attacker action and impact. These implicit entities exhibit limited presence in vulnerability descriptions, which affects the model training. POS solution overcomes this weakness because it relies on grammatical patterns in the sentence and not its frequency of occurrence. In addition, the POS solution outperforms the machine learning solution in identifying explicit entities such as file names (better *Rec.*

Table 2. Accuracy metrics and t-test results for the two approaches on our corpus. "Diff" is the ML minus POS – difference between the two approaches on that metric; "Max" is the maximum value for the two.

		ML		POS				t-test
Label	Metric	Mean	SD	Mean	SD	Diff	Max	$P <$
File	Prec	0.96	0.09	0.72	0.15	0.24	0.96	0.015
	Rec	0.35	0.22	0.96	0.06	-0.60	0.96	0.001
	F1	0.58	0.14	0.81	0.12	-0.24	0.81	0.021
Impact	Prec	0.58	0.08	0.94	0.06	-0.36	0.94	0.001
	Rec	0.79	0.11	0.80	0.11	-0.01	0.80	0.880
	F1	0.67	0.09	0.86	0.08	-0.19	0.86	0.007
NER-Modifier	Prec	0.94	0.02	0.99	0.01	-0.05	0.99	0.003
	Rec	0.82	0.40	0.97	0.02	-0.14	0.97	0.132
	F1	0.96	0.01	0.98	0.01	-0.02	0.98	0.108
Operating System	Prec	0.54	0.40	0.26	0.08	0.28	0.54	0.348
	Rec	0.63	0.40	0.90	0.10	-0.26	0.90	0.377
	F1	0.56	0.39	0.40	0.10	0.16	0.56	0.551
Software	Prec	0.53	0.05	0.71	0.06	-0.17	0.71	0.001
	Rec	0.75	0.03	0.77	0.07	-0.02	0.77	0.509
	F1	0.62	0.04	0.74	0.04	-0.11	0.74	0.003
Attacker	Prec	0.12	0.12	0.94	0.04	-0.82	0.94	0.001
	Rec	0.30	0.30	0.76	0.07	-0.46	0.76	0.001
	F1	0.17	0.17	0.84	0.05	-0.67	0.84	0.001
User	Prec	0.00	0.00	0.48	0.69	-0.48	0.48	0.158
	Rec	0.00	0.00	0.60	0.55	-0.60	0.60	0.040
	F1	0.00	0.00	0.27	0.27	-0.27	0.27	0.054
Version	Prec	0.90	0.05	0.94	0.02	-0.05	0.94	0.070
	Rec	0.97	0.03	0.90	0.03	0.07	0.97	0.006
	F1	0.93	0.04	0.92	0.03	0.01	0.93	0.572
Vulnerability	Prec	0.31	0.11	0.59	0.11	-0.28	0.59	0.004
	Rec	0.62	0.14	0.73	0.06	-0.11	0.73	0.133
	F1	0.42	0.12	0.65	0.08	-0.24	0.65	0.007

and *F1*), software names (better *Prec.*, *Rec.* and *F1*) and versions (better *Prec.*). Therefore, we can conclude that POS tagging approach provides a feasible alternative to ML in security domain without the need for creating and maintaining large corpora.

References

1. Bridges, R.A., et al.: Automatic labeling for entity extraction in cyber security. Computing Research Repository. Available at `http://arxiv.org/abs/1308.4941` (2013)
2. Esuli, A., Sebastiani, F.: SentIWordNet: A publicly available lexical resource for opinion mining. In: Proceedings of the 5th Conference on Language Resources and Evaluation. Genoa, Italy (May 2006)
3. Fellbaum, C.: WordNet: An Electronic Lexical Database. Bradford Books (1998)

4. Finkel, J.R., et al.: Incorporating non-local information into information extraction systems by Gibbs sampling. In: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics. Ann Arbor, MI (June 2005)
5. Joshi, A., et al.: Extracting cybersecurity related linked data from text. In: Proceedings of the 7th IEEE International Conference on Semantic Computing. Irvine, CA (September 2013)
6. Klein, D., Manning, C.D.: Accurate unlexicalized parsing. In: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics. Sapporo, Japan (July 2003)
7. Lab, N.: BRAT annotation tool. `http://brat.nlplab.org/` (2010)
8. Makhoul, J., et al.: Performance measures for information extraction. In: Proceedings of DARPA Broadcast News Workshop. Herndon, VA (March 1999)
9. de Marneffe, M.C., et al.: Generating typed dependency parses from phrase structure parses. In: Proceedings of the International Conference on Language Resources and Evaluation. Genoa, Italy (May 2006)
10. McNeil, N., et al.: PACE: Pattern accurate computationally efficient bootstrapping for timely discovery of cyber-security concepts. Computing Research Repository. Available at `http://arxiv.org/abs/1308.4648` (2013)
11. Mulwad, V., et al.: Extracting information about security vulnerabilities from web text. In: Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology. Lyon, France (August 2011)
12. Roschke, S., et al.: Towards unifying vulnerability information for attack graph construction. In: Proceedings of the 12th International Conference on Information Security. LNCS 5735 (2009)
13. Settles, B.: Biomedical named entity recognition using conditional random fields and rich feature sets. In: Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and Its Applications. Geneva, Switzerland (August 2004)
14. Toutanova, K., Manning, C.D.: Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In: Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora. Hong Kong (October 2000)
15. Urbanska, M., et al.: Structuring a vulnerability description for comprehensive single system security analysis. In: Rocky Mountain Celebration of Women in Computing. Fort Collins, CO, USA (November 2012)
16. Urbanska, M., et al.: Accepting the inevitable: Factoring the user into home computer security. In: Proceedings of the Third ACM Conference on Data and Application Security and Privacy. San Antonio, TX, USA (February 2013)
17. Wallach, H.M.: Conditional random fields: An introduction. CIS Technical Report MS-CIS-04-21, University of Pennsylvania (2004)