Vulnerability Identification and Classification Via Text Mining Bug Databases

Dumidu Wijayasekara, Milos Manic
University of Idaho
Idaho Falls, ID, USA
dumidu.wijayasekara@gmail.com, misko@ieee.org

Miles McQueen
Idaho National Laboratory
Idaho Falls, ID, USA
miles.mcqueen@inl.gov

***Abstract*—As critical and sensitive systems increasingly rely on complex software systems, identifying software vulnerabilities is becoming increasingly important. It has been suggested in previous work that some bugs are only identified as vulnerabilities long after the bug has been made public. These bugs are known as Hidden Impact Bugs (HIBs). This paper presents a hidden impact bug identification methodology by means of text mining bug databases. The presented methodology utilizes the textual description of the bug report for extracting textual information. The text mining process extracts syntactical information of the bug reports and compresses the information for easier manipulation. The compressed information is then utilized to generate a feature vector that is presented to a classifier. The proposed methodology was tested on Linux vulnerabilities that were discovered in the time period from 2006 to 2011. Three different classifiers were tested and 28% to 88% of the hidden impact bugs were identified correctly by using the textual information from the bug descriptions alone. Further analysis of the Bayesian detection rate showed the applicability of the presented method according to the requirements of a development team.**

Keywords— hidden impact bugs; bug database mining; vulnerability discovery; text mining; classification

I. INTRODUCTION

Vulnerabilities in software have always been an important security focus. As control and monitoring systems become dependent on complex software, discovering these software vulnerabilities as early as possible is extremely important. Early identification of vulnerabilities will minimize the time in which the vulnerabilities expose the systems to attack.

Industrial software systems that control and monitor critical and sensitive infrastructure such as the power grid and manufacturing plants are becoming increasingly dependent on complex software systems and communications [1], [2]. Furthermore, as systems become more complex the number of vulnerabilities also increases [3]. While network communication related vulnerabilities are important [4], it has been shown that a significant portion of the vulnerabilities exist in applications and therefore a significant portion of attacks are aimed at the application layer [5], [6].

Hidden Impact Bugs (HIBs) can be defined as vulnerabilities identified as such only after the related bug had been disclosed to the public [7], [8]. These software bugs are disclosed to the public via bug databases and bug fixes, before being identified as having a high security impact and being labeled as vulnerabilities. Thus, even though a bug is known to the community it may not be as quickly fixed by developers, and a fix may not be applied in an appropriately timely fashion by end-users, because the security implication of the bug has not been correctly identified.

It was shown in [8] that a significant portion (32% for Linux and 62% for MySQL) of discovered software vulnerabilities was initially reported as HIBs. It was also shown in [8] that the percentage of HIBs has increased in recent years. Thus, a methodology that identifies these HIBs as they are reported to bug databases will reduce the time critical systems are exposed to these vulnerabilities.

This paper presents a software vulnerability identification methodology using HIBs, that utilizes the textual description of the bugs that were reported to publically available bug databases. The presented methodology utilizes text mining techniques to 1) extract syntactical information of bug reports, 2) compress the information for easier manipulation, and 3) use this information to generate a feature vector which is used for classification. Thus, the presented system is intended to classify bugs as potential vulnerabilities as they are being reported to bug databases, thereby reducing the time software is exposed to attack through the vulnerability.

The presented methodology was tested on Linux bug reports that were reported to the Redhat Bugzilla bug database [9] within the time period from January 2006 to April 2011, and Linux kernel vulnerabilities that were reported in the MITRE CVE [10] database during the same time period. Using the presented text mining steps, a feature vector was generated and different classifiers were used to classify HIBs and regular bugs. Three different commonly used classifiers were used for the classification and 28% to 88% of the HIBs were identified correctly. An evaluation of the classifiers over time, based on the data available at a given time was performed to investigate the usability of such classifiers in real world scenarios. An analysis utilizing the Bayesian detection rate of the classifiers was also performed to further investigate the classification accuracy and the affect of false positives for vulnerability detection.

The rest of the paper is organized as follows. Section II briefly discusses related work in the area. Section III elaborates the proposed vulnerability identification methodology and the text mining process involved. Section IV discusses the experimental setup used in this paper in detail. Finally section V discusses the experimental results and section VI concludes the paper.

II. Related Work

Bug databases are used by software developers to identify and keep track of information about software bugs that were not identified at the time of software release. Developers will utilize these bug reports for different purposes such as improving reliability and improving future requirements [11], [12]. Publically available bug databases enable users to report bugs as they encounter it and search the bug database for bugs they might encounter in the future [13]. Bug databases also keep track of the fixes being released for different bugs and what stage of the resolution process a bug is in. Because different entities with different levels of expertise and requirements report bugs to these databases, the information contained in bug reports is highly noisy and not in standard form [8], [14], [15]. However, this information has been successfully used for various classification purposes [11], [14], [15], [16].

Detection of duplicate bug reports using textual data of bug reports were explored in [17], [18]. A tool for identifying duplicate bug reports in Apache, Eclipse and Linux bug databases was proposed by Wu et al. in [19].

In [14] and [15] Lamkanfi et al. used the textual description of bug reports to classify severity of bugs. In [15] classification algorithms such as Naïve Bayes and Naïve Bayes Multinomial were compared for classifying Eclipse and GNOME bug reports.

Software vulnerability discovery is largely focused on source code analysis. In [20] the authors used machine learning and text mining to analyze the source code to identify vulnerabilities. However, the results were below expectations. Similar work was done in [21] to predict defect-proneness in software. A text mining based approach that analyzes source code was explored to identify mobile device vulnerabilities in [22].

Several commercially available tools for vulnerability identification rely on static code analysis. However, it has been suggested that these tools as well as free tools may have high false positive rates [23]. Furthermore, it has been shown that in order to achieve realistic prediction rates, either multiple static analysis tools must be used or significant manual adjustment of these tools is required [23].

III. Hidden Impact Bugs Classification Methodology

As mentioned earlier, previous work has shown that a significant portion of vulnerabilities are only identified as such after they have been publically disclosed [7], [8]. Thus, although some vulnerabilities were identified as bugs in the code at an earlier time, the actual security impact of these bugs were not detected till later. Therefore, this paper proposes a methodology that utilizes information in bug reports to identify vulnerabilities as they are reported to bug databases. This section will first elaborate on the overall classification process and then discuss the text mining techniques used in detail.

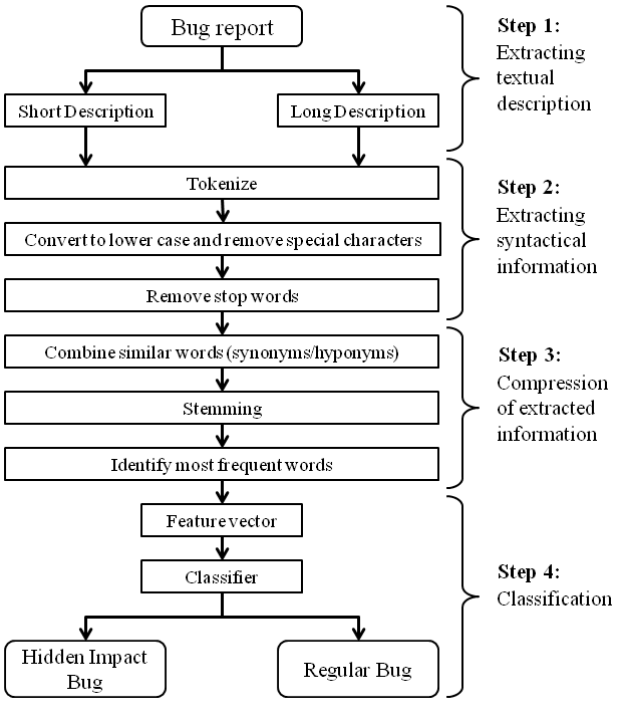

Fig. 1. Identification methodology of vulnerabilities using bug reports

A. The classification process

The presented methodology utilizes the textual description of reported bugs to generate a unique feature vector that will be used by a classifier. The classifier will classify reported bugs as potential vulnerabilities or regular bugs. Fig. 1 illustrates the classification process. The overall process can be separated into four major steps.

In **Step 1** (Fig. 1) the short and long descriptions of the bug report is extracted. The short description is a title provided by the reporter that is around 5-10 words in length. The long description is a more detailed description of the bug which may include how to recreate it, code snippets, memory dumps, etc.

In **Step 2**, the most important and recurring syntactical information is extracted from the short and long descriptions of the bugs. This is performed via text mining techniques which will be discussed later in Section II.B. The syntactical information is extracted in the form of single unique words known as keywords. The extraction process removes words and symbols that might not carry a significant amount of information, and only extracts single words.

In **Step 3**, compression of this extracted information is performed. Text mining techniques are used in this step to identify words that may carry similar information and combine them. This step reduces the feature space which decreases the resource utilization of the process. This step also counts the number of bugs each keyword has appeared in and identifies the most frequently used keywords in the bug descriptions.

Finally, in **Step 4**, extracted set of keywords are used to create a feature space for the bug descriptions. Each dimension of the feature space consists of a set of words that carry similar information. This feature vector can be used by a classifier to perform the final classification. Thus the classifier will classify a given bug as a potential HIB or a regular bug.

B. Text mining process

This section explains the main text mining steps (**Step 2** and **Step 3**) in detail. **Step 2** and **Step 3** consist of three different stages each.

The first main step of the text mining process is extracting syntactical information (**Step 2** in Figure 1). The syntactical information of a textual document can be represented as a *bag-of-words* where we store the number of times each unique keyword (*term*) is found in a bug report. This representation is also known as the *term frequency* representation [24]. Thus a document D_i containing n unique words can be represented as:

$$D_i = \{t_{i1}, t_{i2}, ..., t_{in}\} \quad (1)$$

where, t_{ij} is the number of times term t_j appeared in document i. As the name suggests the bag-of-words representation only takes into account words and disregards numbers and special characters in the text. This is because numbers and special characters carry very little to no information when taken out of context. Similarly, the cases (upper and lower) of the letters in words are also disregarded for the same reason.

Furthermore, frequently occurring words in English language, known as *stop words*, are also removed from the bag-of-words representation. These words include Pronouns such as: "*I, he, she*", Articles such as: "*a, an, the*", Prepositions such as: "*after, to, but*", Conjunctions such as: "*and, but, when*", and other frequently appearing words. Such words also carry very little to no information when taken out of context, and are therefore disregarded.

In order to successfully capture all the information in a set of N documents, the term frequency of all M unique words in the set of N documents must be used. Thus the syntactical information of N documents with M unique words can be expressed using a $N \times M$ matrix:

$$TDM = \begin{bmatrix} D_1 = \{t_{11}, t_{12}, ..., t_{1M}\} \\ .... \\ D_N = \{t_{N1}, t_{N2}, ..., t_{NM}\} \end{bmatrix} \quad (2)$$

This $N \times M$ matrix is called the Term-Document Matrix (TDM).

The main problem faced when using the TDM is, as the number of documents (N) increases, the number of unique words (M) also increases. This results in a large matrix which leads to increased resource usage and higher computational times. Since many of the unique words might not appear in most of the documents, the TDM can be extremely sparse as well.

In order to alleviate these problems and better represent the information in textual documents, further text mining techniques were applied to the problem discussed in this paper. These techniques reduce the dimensionality, M of the TDM by combining words that carry similar information and further removing words that carry little information. This is the second main step of the text mining process, which is compression of the extracted syntactical information (**Step 3** in Fig. 1).

The compression step consists of three sequential stages. The first stage of compression is identifying and combining synonyms. Synonyms are words that have the same meaning or nearly the same meaning as another word. Thus identifying and combining synonyms leads to a reduced dimensionality with very little loss of information. In this paper, the English word database Wordnet [25] was used to identify synonyms. After synonyms are identified, they can be combined to form a single dimension in the TDM. Since similar words are combined to form one dimension, each dimension m_i in the TDM can be represented as a set of r keywords, where the keyword set m_i can be represented as:

$$m_i = \{a_{i1}, a_{i2}, ..., a_{ir}\} \quad (3)$$

where a_{ik} is the k^{th} word in the dimension m_i. Before identifying synonyms the number of words in each dimension, r is 1 for all dimensions.

Let "*word a is a synonym of word b*" be represented as $a \approx b$, then, for the two keyword sets m_i and m_j represented by:

$$m_i = \{a_{i1}, a_{i2}, ..., a_{iA}\} \quad (4)$$

$$m_j = \{b_{j1}, b_{j2}, ..., b_{jB}\} \quad (5)$$

$$m_i = m_j \ if \ any \ a_{ik} \approx b_{jl} \vee b_{jl} \approx a_{ik} \ \forall \ k,l \quad (6)$$

Furthermore, if $m_i = m_j$:

$$m(syn)_{i,j} = m_i \bigcup m_j \quad (7)$$

and m_i and m_j is deleted from the TDM and $m(syn)_{i,j}$ is added to the TDM.

This process is iterated over the complete set of identified words M in the TDM, until there are no more dimensions that satisfy $m_i = m_j$.

After this process the dimensionality of the TDM, M is reduced to M':

$$M' = M - syn \quad (8)$$

where:

$$syn = \sum_{i=1}^{M} (q_i - 1) \qquad \forall q_q > 0 \quad (9)$$

TABLE I. NUMBER OF BUGS REPORTED PER YEAR IN THE REDHAT BUGZILLA [9] DATABASE

Year	Number of bug reports	Average number of bugs reported per day
Prior to 2006	59,819	22.9
2006	15,249	41.8
2007	17,217	47.2
2008	20,817	57.0
2009	26,950	73.8
2010	43,120	118.1
2011 (to April)	17,616	146.8
Other unknown	2108	-
Total	202.896	44.3

where, M is the dimensionality of the TDM and r_i is the number of keywords of each keyword set.

Thus, the use of synonyms effectively reduces the dimensionality of the TDM. Since synonyms are words that carry similar information, the loss of information of this process is minimal.

The second stage in the compression step (**Step 3**, Fig. 1) of the text mining process is deconstructing words into their base forms and combining similar words. The deconstruction of words into their base form is known as stemming. The specific stemming algorithm used in this paper is called Porter stemming [26]. Stemming is capable of deconstructing words that have been transformed, for example by pluralizing or by adding a gerund, into their basic form. This enables identification of transformed words as similar to their base words.

The process of indentifying similar stemmed words and combining them is similar to the process described above for synonyms. Therefore as with identifying and combining synonyms, the dimensionality of the TDM is reduced with minimal loss of information.

The third and final stage of the compression step is identifying the most frequently used keyword sets in the TDM. This is done by counting the number of documents each keyword set appears in, and selecting the keyword sets which appear most often in documents. Typically keyword sets that appear in less than *P%* of the documents are discarded. Although this type of threshold selection reduces the dimensionality of the TDM significantly and identifies words that are most general to the document set, it may also remove words that are important to the classification and retain words that may not contribute to the classification.

IV. EXPERIMENTAL SETUP

This section explains in detail the experimental setup that was used to test the presented vulnerability classification methodology. The presented methodology was tested on a set of bug reports and HIBs for the Linux kernel that were reported in the time period from January 2006 to April 2011.

A. Vulnerability and Bug Databases used

For this paper the MITRE CVE vulnerability database [10] was used to identify HIBs for the Linux kernel. The Redhat Bugzilla bug database [9] was used as the bug database. All the bugs and vulnerabilities explored in this paper are within the time period from January 2006 to April 2011.

TABLE II. NUMBER OF SELECTED REGULAR BUGS AND HIDDEN IMPACT BUGS FROM EACH YEAR

Year	Number of regular bugs	Number of hidden impact bugs found
2006	642	3
2007	725	12
2008	876	21
2009	1135	10
2010	1,819	25
2011 (to April)	803	2
Total	6000	73

In order to identify vulnerabilities that are most applicable and most relevant, only vulnerabilities that affected 1) multiple processors, 2) multiple distributions and 3) Linux kernel 2.6 and above, were considered in this paper. HIBs were identified as vulnerabilities that had at least 2 weeks of impact delay, where the impact delay was defined as the time from the public disclosure of the bug via a patch to the time a CVE was assigned to the vulnerability in the MITRE database.

Out of the Linux kernel vulnerabilities reported from January 2006 to April 2011 in the MITRE CVE database, 403 vulnerabilities were identified as most relevant using the rules mentioned above. Out of these 129 (39%) showed an impact delay of at least 2 weeks [8]. Unfortunately, out of the 129 HIBs identified, only 73 had accessible bug reports in the Redhat Bugzilla bug database.

As of 2011-4-30 the Redhat Bugzilla database contained 202,896 entries. Table I shows the distribution of bugs per year and the mean number of bugs reported per day each year in the Redhat Bugzilla bug database. The “Other Unknown” bugs in Table I refer to bugs that were not considered due to no report date, no textual descriptions or due to denied access.

B. Classification subset

In order to test the presented vulnerability classification methodology, a set of Redhat Linux bugs, containing two classes: regular bugs and HIBs were constructed.

The MITRE CVE vulnerability database reports the bugs associated with each vulnerability. This information was used to extract bugs in the Redhat Bugzilla bug database that were associated with the identified vulnerabilities. Therefore for the final classification and testing the set of 73 identified HIBs that had accessible bug reports in the Redhat Bugzilla bug database were used. These bugs constitute the HIB class.

The regular bug class contained 6000 randomly selected bugs reported from January 2006 to April 2011 that were not identified as vulnerabilities. Since the number of bugs reported per each year is different for each year (see Table I), in order to avoid misrepresenting any year, the random set was constructed to reflect proportion of bugs reported for each year. However, it is important to note that the regular bug class may contain bugs that are yet to be identified as vulnerabilities, and the classifiers may be negatively affected by training on these examples.

TABLE III. CONFUSION MATRIX

		Classified as	
		HIB	Regular
Actual Class	HIB	True Positives (TP)	False Negatives (FN)
	Regular	False Positives (FP)	True Negatives (TN)

TABLE IV. CLASSIFICATION RESULTS FOR THE CLASSIFIERS TESTED

Classifier	TP Rate	TN Rate
Naïve Bayes (NB)	0.88	0.46
Naïve Bayes Multinomial (NBM)	0.78	0.90
Decision Tree (DT)	0.28	0.99

TABLE V. BAYESIAN DETECTION RATE OF THE CLASSIFIERS TESTED

Classifier	Bayesian Detection Rate	Number of Bugs Classified Per Day (2011)
Naïve Bayes (NB)	0.02	80.3
Naïve Bayes Multinomial (NBM)	0.09	16.5
Decision Tree (DT)	0.40	1.2

Table II shows the number of HIBs that were identified for each year and the number of regular bugs selected from each year for the classification process.

C. Construction of TDM

The Term-Document Matrix (TDM) was constructed using the short and long descriptions of the bug reports. The short and long descriptions of the bugs were treated separately, meaning the text mining process was applied to words extracted from the short description and the long description separately. This is because a word in the short description may carry different information compared to the same word in the long description.

The percentages P for selecting the keywords appearing most frequently in bug reports were set at P_{SDESC} = 1% and P_{LDESC} = 3%. These numbers were selected somewhat arbitrarily and the same thresholds were used to test each classifier. Although this type of selection is sub-optimal for classification, it is sufficient for demonstrating the vulnerability classification methodology described in this paper.

D. Classifiers tested

Three different classifiers were tested on the textual information extracted from bug reports: 1) Naïve Bayes (NB), 2) Naïve Bayes Multinomial (NBM), and 3) Decision Tree (DT). These classifiers were chosen because of their high interpretability, fast learning phase and capability of classifying highly multi dimensional data. The selected classifiers will classify a given bug as an HIB or regular bug.

NB [27] and NBM [28] use the Bayes theorem of conditional probability to calculate the conditional probability of a class given a set of keywords. The NB classifier only considers the presence or absence of a keyword in a document, whereas the NBM classifier considers the number of times each keyword occurred in each document [28]. DT use information entropy and information gain of each dimension to generate a set of optimal dimensions to classify the dataset on. The particular version of DT used in this paper, called C4.5 [29], also incorporates heuristic methodologies to reduce the number of nodes in the final tree.

V. EXPERIMENTAL RESULTS

The classification was performed using 10-fold cross validation. Classification results are shown in terms of True Positives (TP) and True Negatives (TN) as shown in Table III.

The overall classification results are shown in Table IV. Naïve Bayes classifier showed the best TP rate (88%), however, the TN rate was low. Similarly the decision tree had a very low TP rate (28%) but the highest TN rate (99%). Naïve Bayes Multinomial showed a higher TP rate as well as higher TN rate. Although these results may be relatively low, even the lowest TP rate (28%) is far better than a random guess (1.2%).

According to Table II, the number of reported bugs and the number of identified vulnerabilities have been increasing each year. In order to evaluate the usability of the classifiers in this real world scenario, the performance of each classifier was measured across time, based on the data available at a given moment in time. For this analysis, cumulative bugs reported and HIBS found at the start of each year was selected (Table II).

Figs. 2, 3 and 4 plot the yearly classification results for Naïve Bayes, Naïve Bayes Multinomial and Decision Tree classifiers respectively. As expected, the TP rate increases with time. This is because the size of the HIB set is increasing. Therefore, as more HIBs are correctly classified as vulnerabilities, the classifiers are able to adapt to the new data and benefit from these newly discovered HIBs.

Since the size of the Regular bug set is very large compared to the HIB set (6000 vs. 73), even a low rate of FP may result in an overwhelming amount of bugs incorrectly classified as HIBs. In order to identify this problem, Bayesian detection rate is used which is the probability that an instance classified as true, is actually true [30]. Thus, a higher Bayesian detection rate means that a lower percentage of regular bugs were improperly classified as vulnerabilities and therefore a software maintenance team will have to sort through fewer regular bugs to find those which are actual vulnerabilities. Therefore, a higher Bayesian detection rate is preferred. The Bayesian detection rate can be calculated using the following equation:

$$BayesianDetectionRate = \frac{TP}{(TP + FP)} \tag{10}$$

Furthermore, using the average number of bugs reported per day, the number of bugs that will be classified as HIBs by the classifier can be calculated as:

$$\frac{TP + FP}{(TP + FP + TN + FN)} \times BugsReportedperDay \tag{11}$$

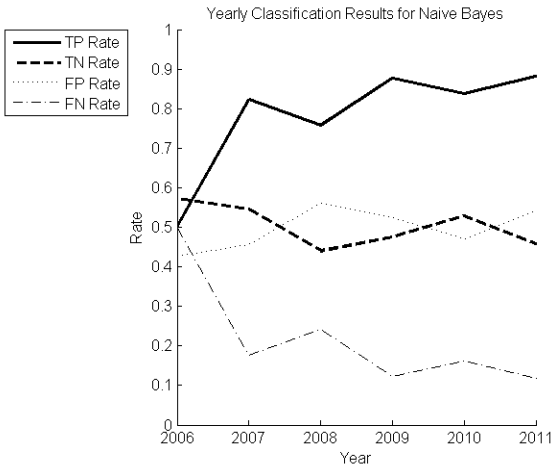

Fig. 2. Classification results for Naïve Bayes Classifier

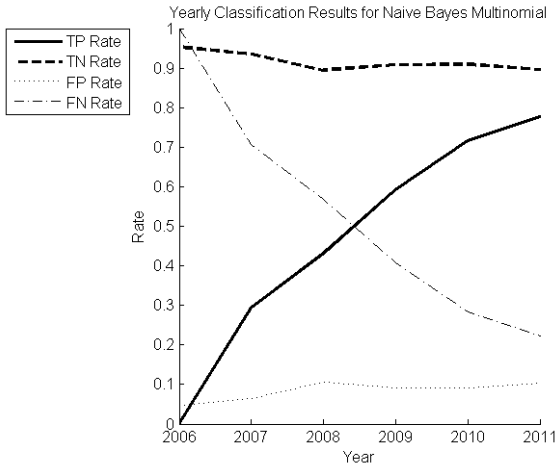

Fig. 3. Classification results for Naïve Bayes Multinomial Classifier

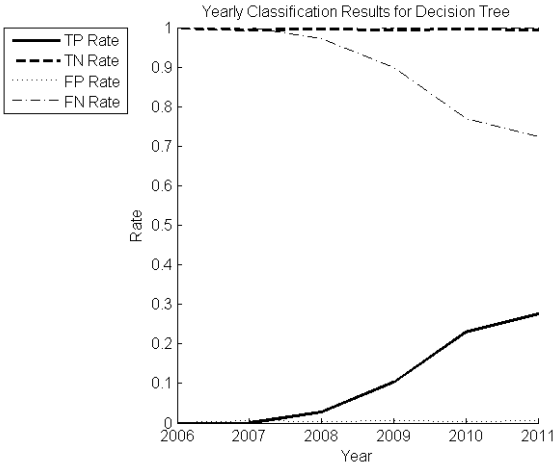

Fig. 4. Classification results for Decision Tree Classifier

The Bayesian detection rate and the number of bugs that will be classified as HIBs by each classifier for a given day in 2011 is shown in Table V. These results show that although the DT had a low TP rate more than a third of bugs that are classified as vulnerabilities are actual vulnerabilities. This also leads to less than 2 bugs being classified as vulnerabilities each day in 2011 where over 145 bugs were reported to the bug database each day (see Table I).

Similarly, NBM classifier reported a Bayesian detection rate of 0.09, which means that just under one out of 10 bugs that are classified as a vulnerability is an actual vulnerability. Due to the very high FP rate of the NB classifier, only 1 out of 50 bugs that are classified as vulnerabilities are actual vulnerabilities.

The experimental results show that the tested classifiers were capable of identifying HIBs in the Linux bug database using the textual information stored in the bug reports. Although the classification accuracy was relatively low, it was shown to be at least an order of magnitude better than a random guess. Furthermore, as mentioned above, the regular bug set may contain HIBs that are yet to be correctly identified as vulnerabilities. These HIBs may be contained in the false positives of the classifiers.

It should be understood that the number of keywords in the final TDM play an important role in the classification results. Although the tests presented in this paper were carried out using the same set of keywords for each classifier, some classifiers may produce better results for different sets of keywords. Furthermore, important keywords that contribute to the classification may not be included when the final TDM is generated.

VI. Conclusion

This paper presented a methodology for identifying vulnerabilities in software systems for Hidden Impact Bugs (HIBs) using textual information from publically available bug databases. The presented methodology uses text-mining techniques to extract syntactical information from bug reports, and generate a feature vector that can be used by classification algorithms. Naïve Bayes, Naïve Bayes multinomial, and Decision tree classification was tested on a set of bugs extracted from the Redhat Bugzilla bug database and HIBs identified for the Linux Kernel. The tested classifiers were able to correctly classify 28% to 88% of the HIBs.

An analysis of the classifiers over time using the cumulative data gathered up to that time showed the ability of the classifiers to adapt to new data as well as the real world usability of such a classification methodology. Further analysis on the Bayesian detection rate of the classifiers showed that the number of bugs that will be classified as vulnerabilities per day given the results of each classifier (from less than 2 to over 80). This information can be used by developers to select the optimal classifier, given the number of bugs that can be handled by a software maintenance team.

As future work the feature extraction process will be enhanced to incorporate information from already identified HIBs. Further improvements can be achieved by identifying keywords that contribute to the classification more and including these dimensions in the feature vector. Multiple classifiers will be used in series or parallel to further increase the classification accuracy. The feature vector can be enhanced

by incorporating other dimensions of the bug reports and attributes of the source code itself and other aspects of the software development process.

REFERENCES

[1] M. McQueen, "Software and human vulnerabilities," in *Proc. IEEE. Int. Conf. of the Industrial Electronics Society, (IECON)*, pp. 1-85, Nov. 2010.

[2] C. Neureiter, G. Eibl, A. Veichtlbauer, D. Engel, "Towards a framework for engineering smart-grid-specific privacy requirements," in *Proc. IEEE. Int. Conf. of the Industrial Electronics Society, (IECON)*, pp. 4803-4808, Nov. 2013.

[3] H. Isakovic, A. Wasicek, "Secure channels in an integrated MPSoC architecture," in *Proc. IEEE. Int. Conf. of the Industrial Electronics Society, (IECON)*, pp. 4488-4493, Nov. 2013.

[4] N. Moreira, A. Astarloa, U. Kretzschmar, "SHA-3 based Message Authentication Codes to secure IEEE 1588 synchronization systems," in *Proc. IEEE. Int. Conf. of the Industrial Electronics Society, (IECON)*, pp. 2323-2328, Nov. 2013.

[5] Veracode. (Apr 8, 2013). State of Software Security Report Volume 5. [Online], Avalable: http://www.veracode.com/resources/state-of-software-security.

[6] Cenzic. (2014). Application Vulnerability Trends Report. [Online], Available:http://www.cenzic.com/downloads/Cenzic_Vulnerability_Report_2014.pdf

[7] J. Arnold, T. Abbott, W. Daher, G. Price, N. Elhage, G. Thomas, A. Kaseorg, "Security Impact Ratings Considered Harmful," in *Proc. of the 12th Conf. on Hot Topics in Operating Systems, (USENIX)*, May 2009.

[8] D. Wijayasekara, M. Manic, J. L. Wright, M. McQueen "Mining Bug Databases for Unidentified Software Vulnerabilities," in *Proc of the 5th Intl. IEEE Intl. Conf. on Human System Interaction, (HSI)*, June, 2012.

[9] Redhat, Inc. (1 May 2014), *Redhat Bugzilla Main Page* [Online]. Available: https://bugzilla.redhat.com/.

[10] The MITRE Corporation (1 May 2014), *Common Vulnerabilities and Exposures (CVE)* [Online]. Available: http://cve.mitre.org/.

[11] A. J. Ko, B. A. Myers, D. H. Chau, "A Linguistic Analysis of How People Describe Software Problems," in *Proc. of the 2006 IEEE Symp. on Visual Languages and Human-Centric Computing (VL/HCC 2006)*, pp. 127–134, Sep. 2006.

[12] M. F. Ahmed, S. S. Gokhale, "Linux Bugs: Life Cycle and Resolution Analysis," in *Proc of The 8th Int. Conf. on Quality Software (QSIC '08)*, Aug. 2008, pp.396–401.

[13] J. Noll, S. Beecham, D. Seichter, "A Qualitative Study of Open Source Software Development: the OpenEMR Project," in *Proc of the Int. Symp. on Empirical Software Engineering and Measurement (ESEM '11)*, pp. 30–39, Sep. 2011.

[14] A. Lamkanfi, S. Demeyer, E. Giger, B. Goethals, "Predicting the severity of a reported bug," in *Proc. of the 7th IEEE Working Conf. on Mining Software Repositories (MSR 2010)*, pp. 1–10, May 2010.

[15] A. Lamkanfi, S. Demeyer, Q. D. Soetens, T. Verdonck, "Comparing Mining Algorithms for Predicting the Severity of a Reported Bug," in *Proc. of the 15th European Conf. on Software Maintenance and Reengineering (CSMR)*, pp.249–258, Mar. 2011.

[16] P. Bhattacharya, I. Neamtiu, C. R. Shelton, "Automated, highly-accurate, bug assignment using machine learning and tossing graphs," in *The Journal of Systems and Software*, vol. 85, pp. 2275-2292, 2012.

[17] P. Runeson, M. Alexandersson, O. Nyholm, "Detection of Duplicate Defect Reports Using Natural Language Processing," in *Proc. of the 29th Int. Conf. on Software Engineering (ICSE 2007)*, pp. 499–510, May 2007.

[18] T. Prifti, S. Banerjee, B. Cukic, "Detecting Bug Duplicate Reports through Local References," in *Proc of the 7th Int. Conf. on Predictive Models in Software Engineering (PROMISE '11)*, pp. 8:1–8:9, Sep. 2011.

[19] L. Wu, B. Xie, G. Kaiser, R. Passonneau, "BugMiner: Software Reliability Analysis Via Data Mining of Bug Reports," in *Proc. Int. Conf. on Software Engineering and Knowledge Engineering, (SEKE)*, pp. 95–100. 2011.

[20] F. Yamaguchi, F. Lindner, K. Rieck, "Vulnerability Extrapolation: Assisted Discovery of Vulnerabilities using Machine Learning," in *Proc. of the 5th USENIX Workshop on Offensive Technologies (WOOT)*, USENIX, Aug. 2011.

[21] L. Li, H. Leung, "Mining Static Code Metrics for a Robust Prediction of Software Defect-Proneness," in *Proc of the 2011 Int. Symp. on Empirical Software Engineering and Measurement, (ESEM '11)*, pp. 207–214, Sep. 2011.

[22] A. Hovsepyan, R. Scandariato, W. Joosen, J. Walden, "Software Vulnerability Prediction using Text Analysis Techniques," in *Proc. of MetriSec'12*, Sep. 2012.

[23] A. Austin, L. Williams "One Technique is Not Enough: A Comparison of Vulnerability Discovery Techniques," in *Proc. of the 2011 Int. Symp. on Empirical Software Engineering and Measurement (ESEM '11)*, pp. 97–106, Sep. 2011.

[24] C. A. Martins, M. C. Monard, E. T. Matsubara, "Reducing the Dimensionality of Bag-of-Words Text Representation Used by Learning Algorithms," in *Proc of 3rd IASTED International Conference on Artificial Intelligence and Applications*, pp. 228–233, 2003.

[25] C. Fellbaum, *WordNet: An Electronic Lexical Database*, Cambridge, MA: MIT Press, 1998.

[26] M. F. Porter, "An algorithm for suffix stripping," in *Program*, vol. 14, no. 3, pp. 130−137, 1980.

[27] Q. B. Duong, E. Zamai, K. Q. Tran Dinh, "Confidence estimation of feedback information using dynamic bayesian networks," in *Proc. IEEE. Int. Conf. of the Industrial Electronics Society, (IECON)*, pp. 3733-3738, Oct. 2012.

[28] A. McCallum, K. Nigam. "A comparison of event models for naive bayes text classification," in *Proc. of AAAI-98 workshop on learning for text categorization*, vol. 752, 1998.

[29] J. R. Quinlan, *C4.5: Programs for machine learning*. Vol. 1. Morgan kaufmann, 1993.

[30] S. Axelsson, "The base-rate fallacy and the difficulty of intrusion detection," in *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 3, pp. 186–205, Aug. 2000.