Mining Bug Databases for Unidentified Software Vulnerabilities

Dumidu Wijayasekara, Milos Manic
University of Idaho
Idaho Falls, ID, USA
wija2589@vandals.uidaho.edu, misko@ieee.org

Jason L. Wright, Miles McQueen
Idaho National Laboratory
Idaho Falls, ID, USA
jlwright@ieee.org, miles.mcqueen@inl.gov

***Abstract*— Identifying software vulnerabilities is becoming more important as critical and sensitive systems increasingly rely on complex software systems. It has been suggested in previous work that some bugs are only identified as vulnerabilities long after the bug has been made public. These vulnerabilities are known as hidden impact vulnerabilities. This paper discusses existing bug data mining classifiers and present an analysis of vulnerability databases showing the necessity to mine common publicly available bug databases for hidden impact vulnerabilities.**

We present a vulnerability analysis from January 2006 to April 2011 for two well known software packages: Linux kernel and MySQL. We show that 32% (Linux) and 62% (MySQL) of vulnerabilities discovered in this time period were hidden impact vulnerabilities. We also show that the percentage of hidden impact vulnerabilities in the last two years has increased by 53% for Linux and 10% for MySQL.

We then propose a hidden impact vulnerability identification methodology based on text mining classifier for bug databases. Finally, we discuss potential challenges faced by a development team when using such a classifier.

***Index Terms*— Hidden impact vulnerabilities, Bug database mining, Vulnerability discovery, Classifier**

I. INTRODUCTION

Software vulnerabilities are an increasing security focus as critical and sensitive systems which operate critical infrastructure become increasingly dependent on complex software systems. Discovering these software vulnerabilities as early as possible, at every stage of the software lifecycle, is therefore of extreme importance in order to minimize the time in which the vulnerabilities expose the systems to attack. This includes the quicker and more effective identification of which bugs may also be vulnerabilities.

In [1], Arnold et al. defined *hidden impact vulnerabilities* as those vulnerabilities identified as such only after the related bug had been disclosed to the public. 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.

Publicly available bug databases store and track information about software bugs. Information contained in bug reports is highly noisy and not in standard form [2], [3]. However, this information has been successfully used for some classification purposes [2], [3], [4].

This paper first extends the work of Arnold [1] and clearly demonstrates, empirically, that there is a need for improved identification of which bugs are also vulnerabilities. Then we address the feasibility and elaborate on the difficulties of mining bug databases for discovery of potential hidden impact vulnerabilities. The Linux kernel and MySQL bug databases were chosen for analysis because they have been deployed for many years, have extensive bug and vulnerability databases, and their source code is available for future use in our classification efforts.

Linux kernel vulnerabilities that were reported in the MITRE CVE [5] database, during the time period from January 2006 to April 2011 were analyzed. This analysis showed a large portion of the most significant vulnerabilities were hidden impact vulnerabilities and this number has increased in the last two years.

A similar analysis of MySQL vulnerabilities reported in the MITRE CVE database was conducted for the time period January 2006 to April 2011. Similar to the Linux kernel, the proportion of hidden impact vulnerabilities was significant and the proportion has increased in the last two years for MySQL as well.

After the empirical analysis of hidden impact vulnerabilities, the practical difficulties of mining bug databases are evaluated in a case study using the Redhat Bugzilla bug database [6]. A vulnerability identification methodology that utilizes text mining techniques to extract information from bug databases and uses machine learning techniques to identify vulnerabilities from this extracted information is then described.

This manuscript has been authored by Battelle Energy Alliance, LLC under Contract No. DE-AC07-05ID14517 with the U. S. Department of Energy. The United States Government retains a nonexclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

The rest of the paper is organized as follows. Related work on bug database mining and vulnerability discovery is discussed in section II. Section III provides an analysis of hidden impact vulnerabilities in the Linux kernel and the MySQL application. Section IV analyzes a publicly available bug database for use in data mining hidden impact vulnerabilities. Section V proposes a hidden impact vulnerability discovery tool and problems associated with such a system. Section VI summarizes the conclusions and provides a brief discussion of our future research.

II. Related Work

This section highlights previous studies into different methods of vulnerability discovery and mining bug databases.

A. Vulnerability discovery

Yamaguchi et al. used machine leaning and text mining techniques to discover vulnerabilities in source code [7]. However, the classification results were below expectations [7]. Similarly, Li and Leung also used machine learning techniques to identify software defects in source code [8].

Many previous studies on vulnerability discovery focused on static code analysis and static code analysis tools. However, it has been shown that there are no universal static analysis tools and static analysis by itself does not provide satisfactory results for vulnerability discovery [9], [10], [11]. The existing tools are also very difficult to use because of the large size of software distributions [12]. Schumacher et al. showed the value of gathering information from vulnerability databases to aid the discovery of vulnerabilities in software [13]. In [14] Torri et al. evaluated 10 free and open source static analysis tools on embedded C programs. Torri et al. found that while the results were very poor, even the best performing tool needed to be tweaked extensively to produce good results, and therefore, this approach was impractical for use in software development and vulnerability discovery [14]. Similar results were shown in [11] and [15].

Zitser et al. tested five static analysis tools on three open source programs [16]. Low detection rates were reported for most of the tools while the best performing tools reported very high false positive rates (false alarm for every 12 to 46 lines of source code) [16].

In [8], Li and Cui compared 7 free and open source static analysis tools and concluded that each by itself did not provide a satisfactory discovery of all vulnerabilities. Thus, it was proposed that a variety of tools be used to compensate for the deficiencies of each tool [8].

Austin and Williams showed that no single technique was able to discover every type vulnerability by itself and therefore, a combination of methods may be the optimal means of vulnerability discovery [10].

B. Bug database mining

Previous studies have shown that the textual data contained in bug reports may carry important information that can help developers in the bug triaging process. Previous work on bug database mining focuses on three main problems: 1) assigning the correct person to fix a bug, 2) finding duplicate bug reports and 3) assigning the correct severity to a reported bug.

In [17], [18] and [19], the authors used text mining to assign the correct person to fix a bug. The correct person can be a developer whose expertise is in that area, or a developer who is responsible for the affected code. In [17] Cubranic and Murphy used Naive Bayes to classify bugs contained in the Eclipse bug database. Anvik et al. used a number of classification techniques to classify bugs in the Eclipse and Firefox databases [18]. In [19], Jeong et al. used a Markov model for the same bug databases and showed better classification accuracy.

Detection of duplicate bug reports is explored in [20], [21], [22] and [23]. Runeson et al. used vector space and cosine similarity measures to find redundant bugs in a Sony Ericsson mobile bug database [20]. In [21], Wang et al. used similarity measures to detect potential duplicate bugs for Eclipse and Firefox bug databases. Wu et al. [23] also proposed a tool for detection of duplicate bugs in Apache, Eclipse and Linux bug databases.

In [2] and [3] Lamkanfi et al. used the textual description of bug reports to classify severity of bugs. In [2] Lamkanfi et al. classified Eclipse, GNOME and Mozilla bugs into three classes of severity using Naive Bayes classifier. In [3] Lamkanfi et al. compared classification algorithms for classifying Eclipse and GNOME bug severity.

However, neither previous studies in vulnerability discovery nor bug database mining focused on discovery of hidden impact vulnerabilities.

III. Hidden Impact Vulnerability Analysis

In this section an analyses of hidden impact vulnerabilities for the Linux kernel and the MySQL database server are presented. It is shown that a significant portion of vulnerabilities are hidden impact vulnerabilities and the number of hidden impact vulnerabilities has, if anything, increased in recent years.

In [1], Arnold et al. defined hidden impact vulnerabilities as those vulnerabilities identified some time after the related bug has been disclosed to the public. This bug disclosure can be via a patch which has been made available to the public or a publicly accessible bug report. The importance of these vulnerabilities, as elaborated in [1] is twofold. First, it is easier for an attacker to use this disclosed information to discover a potentially high impact exploit. Second, even though a patch is available, systems may be at risk, because system administrators tend not to apply lower severity patches that are released for a system. This study focuses on the feasibility of using the disclosed information, in the form of a bug report, to discover a vulnerability before an attacker can take advantage of it.

Hidden impact vulnerabilities for Linux kernel and MySQL database server are analyzed in this section. For each software package, vulnerabilities were divided into two groups depending on when they were first reported: time period from the 1st of January 2006 to the 31st of December 2008, which will be called the *first time period* and the time period from

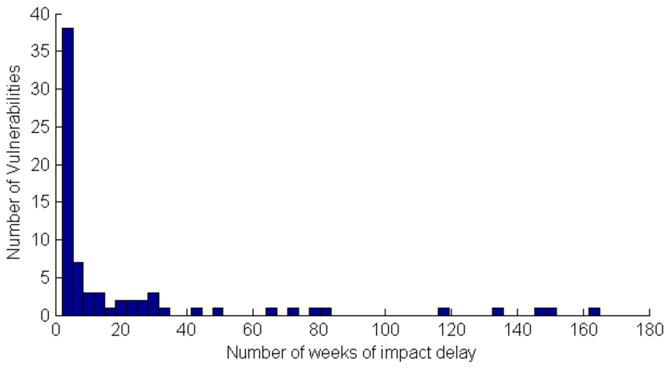

Fig. 1. Number of hidden impact vulnerabilities by impact delay for Linux kernel (January 2009 to April 2011)

TABLE I. HIDDEN IMPACT VULNERABILITIES (LINUX KERNEL)

	2006 Jan. - 2008 Dec. (First time period)	2009 Jan. - 2011 Apr. (Second time period)	Total
Total	218	185	403
At least 2 weeks of impact delay	56 (25.69%)	73 (39.46%)	129 (32.01%)
At least 4 weeks of impact delay	38 (17.43%)	55 (29.73%)	93 (23.08%)
At least 8 weeks of impact delay	31 (14.22%)	29 (15.68%)	60 (14.99%)

the 1st of January 2009 to the 30th of April 2011 which will be called the *second time period*.

A. *Linux Kernel Vulnerability Analysis*

In their study Arnold et al. used a database of Linux kernel vulnerabilities for the first time period (i.e. from the 1st of January 2006 to the 31st of December 2008). For this time period the Linux kernel had 218 vulnerabilities reported out of which 56 (25.69%) had an impact delay of at least 2 weeks. Impact delay was defined as the time from the public disclosure of the bug in the form of a patch to the time a CVE was assigned to the bug because it had now been identified as a vulnerability. It was also shown that for any given day in the time period there was an average of 8.5 hidden impact vulnerabilities present that affected the Linux kernel.

The number of reported vulnerabilities in software has been increasing over the past few years [5], [24]. In order to evaluate whether the number of hidden impact vulnerabilities has also increased over time, a similar analysis was performed for Linux kernel vulnerabilities for the second time period (i.e. from the 1st of January 2009 to the 30th of April 2011). For this analysis specific rules were applied to the vulnerability database downloaded from [5]. Vulnerabilities that affected 1) multiple processors, 2) multiple distributions and 3) Linux kernel 2.6 and above, were selected for the vulnerability database for the time period. Vulnerabilities that affected only a single processor were excluded because these vulnerabilities affected only a small subset of users and it is difficult to identify whether they were caused by a kernel issue. Similarly, vulnerabilities that affected only one distribution were excluded because there is no way of clarifying if the vulnerability was due to a kernel issue. Vulnerabilities that affected Linux kernel 2.6 and above were selected because it was the latest version available in 2006. These rules also seem

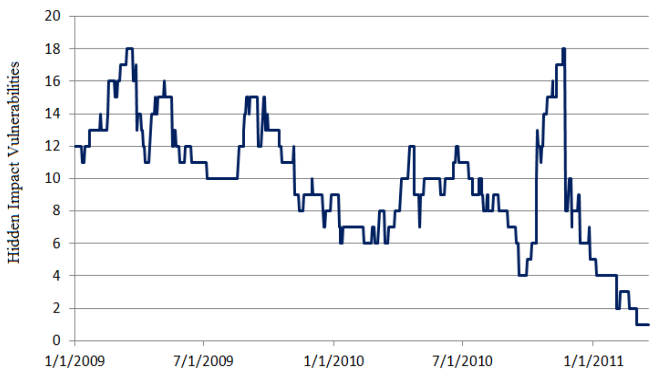

Fig. 2. Number of hidden impact vulnerabilities that existed per day for the Linux kernel (January 2009 to April 2011)

to match the rules applied in [1]. Thus the vulnerability database contained 185 vulnerabilities for the second time period, which is a 15% reduction from the first time period. However, the number of vulnerabilities with at least 2 weeks of impact delay increased to 73 (39.46%). Fig. 1 shows the number of hidden impact vulnerabilities with different impact delays. Table I shows the number of vulnerabilities with at least 2, 4 and 8 weeks of impact delay for the two time periods.

Further, on any given day, there were 9.8 hidden impact vulnerabilities in existence on average during the second time period. Fig. 2 shows the number of hidden impact vulnerabilities that existed on each day for the second time period.

Thus, the number of hidden impact vulnerabilities in the Linux kernel has increased in both percentage and magnitude for the 2009 to 2011 time period. Furthermore, the average number of hidden impact vulnerabilities in existence per each day has also increased for the same time period.

B. *MySQL Vulnerability Analysis*

To expand on the knowledge gained from examining a single product (the Linux kernel), the MySQL database server was analyzed. Like Linux, MySQL has a public database of bugs and a significant number of vulnerabilities in the MITRE CVE database.

Using the same criteria as discussed in Section III.A for the first time period, there were 37 vulnerabilities in the MITRE CVE database out of which 22 (59.5%) had an impact delay of

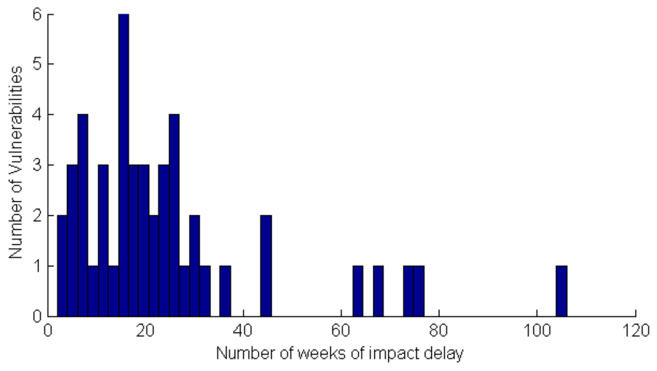

Fig. 3. Number of hidden impact vulnerabilities by impact delay for MySQL (December 2003 to April 2011)

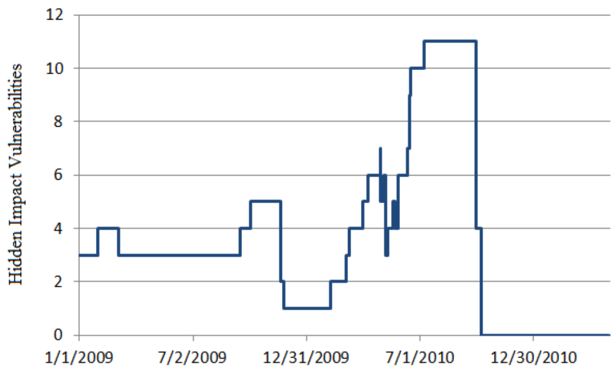

Fig. 4. Number of hidden impact vulnerabilities that existed per day for the MySQL database (January 2009 to April 2011)

TABLE III. HIDDEN IMPACT VULNERABILITIES (MYSQL)

	2006 Jan. - 2008 Dec. (First time period)	2009 Jan. - 2011 Apr. (Second time period)	Total
Total	37	29	66
At least 2 weeks of impact delay	22 (59.46%)	19 (65.52%)	41 (62.12%)
At least 4 weeks of impact delay	21 (56.76%)	19 (65.52%)	40 (60.62%)
At least 8 weeks of impact delay	17 (45.95%)	16 (55.17%)	33 (50%)

at least 2 weeks (see Table II). An average of 3.45 hidden impact vulnerabilities affected the MySQL database server per day for the same time period.

For the second time period, 29 vulnerabilities were reported and 19 (65.5%) of these were hidden impact vulnerabilities that had an impact delay of at least 2 weeks. Although the number of hidden impact vulnerabilities has not increased in absolute terms, it has increased percentagewise in the 2009 to 2011 time period.

Fig. 3 shows the number of vulnerabilities by impact delay for the MySQL database server. Comparing Fig. 1 and Fig. 3 shows that the median impact delay time for MySQL is much higher (11 weeks for Linux and 20 weeks for MySQL). Also, the distribution is of a different shape which may reflect the different priorities of the developers of the two projects.

Finally, Fig. 4 shows the number of hidden impact vulnerabilities on a given day for MySQL for the time period from January 2009 to April 2011. At any given day during the second time period, on average there existed 3.75 hidden impact vulnerabilities for the MySQL database server.

Thus, similar to Linux, MySQL hidden impact vulnerabilities account for a significant portion of the total number of vulnerabilities and the percentage of hidden impact vulnerabilities has increased in the second time period.

IV. EVALUATION OF BUG DATABASES FOR USE IN DATA MINING FOR VULNERABILITIES

Bug databases for software are kept in order to keep track of the bugs existing in the software. Publicly available bug databases benefit from information provided by users with a diverse set of technical backgrounds as well as programmers and developers [25]. These bug databases allow developers to identify previously unforeseen bugs in the software and at the same time users can track the resolution process of each bug. It has been shown that these databases are extremely useful in increasing the quality and reliability of the software as well as containing vital information that can be used for various purposes such as improving future design requirements [4], gathering vital feedback from users [25], and improving software reliability [26], [27].

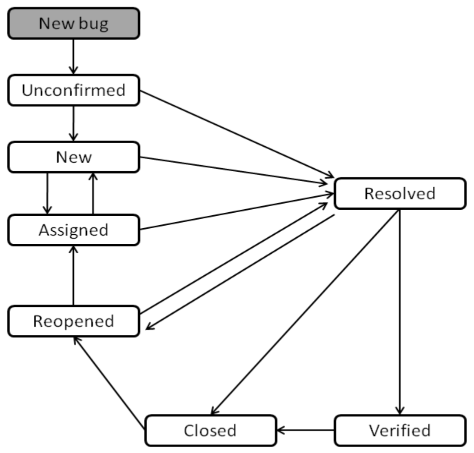

Fig. 5. Typical life cycle of a Bugzilla bug.

In this section bug reports from Redhat Bugzilla database are analyzed. The Redhat Bugzilla database was selected because 1) it is one of the most extensive bug databases available, 2) all other Bugzilla bug databases generally follow the same format, 3) most of the Linux vulnerabilities examined in this paper are associated with bugs in the Redhat Bugzilla database, 4) at the time of writing the paper, Linux Kernel Bugzilla database [28] was restricted from public access due to a security breach. Although the Redhat Bugzilla database "*is not an avenue for technical assistance or support, but simply a bug tracking system*" [6], it has been shown that certain details in the bug reports can be used for various forms of classification as mentioned in Section II.A [2], [3], [4].

A. *Bug Reports and Bug Life Cycle*

After a bug is reported, it is reviewed and the reported bug is assigned a bug ID, which is a unique identifier and enters the bug resolution process. Fig. 5, shows the typical life cycle of a bug after it is reported.

When a bug is reported, the reporter can assign as many parameters to the bug report as he or she sees fit. These parameters include terms such as severity, priority, product, component and keywords. During the life cycle of the bug, these parameters may be changed according to its nature and severity. Apart from these set parameters, the person who reports the bug must provide a title for the bug which is a short description of the bug, and a comment which is a longer description of the bug and should describe the bug in more detail. The long description may include code snippets, how to recreate the bug, the specifications of the hardware setup etc., which are meant to allow the developer to more easily identify and rectify the bug.

The status of the bug changes according to the position of the bug in the life cycle, thus allowing users to be informed on the progress of the bug. Further, comments can be added by

TABLE III. NUMBER OF REPORTED BUGS PER YEAR IN THE REDHAT BUGZILLA [6] DATABASE

Year	Number of bug reports	Number of bugs per day
From Nov.1998	336	5.5
1999	3,788	10.4
2000	5,846	16
2001	7,839	21.5
2002	9,200	25.3
2003	8,497	23.3
2004	11,951	32.7
2005	12,428	34
2006	15,283	41.9
2007	17,263	47.3
2008	20,916	57.3
2009	27,052	74.1
2010	43,301	118.6
to April 2011	19,185	139
Unknown	11	
Total	**202.896**	**44.5**

users and administrators to convey the progress and development of the bug fix or other relevant facts.

As of 2011-4-18 the Redhat Bugzilla database contained 202,896 entries. The first bug which is a test bug report was added to the database on 1998-11-1. Table III shows the distribution of bugs per year and the mean number of bugs per day in the Redhat Bugzilla [6] database. The number of bugs reported has been increasing (see Fig. 6 and 7), which might be due to the surprising fact that mature releases of the same software tend to have more bugs reported [27].

The main problem with such bug reports is that most of the parameters of the bug are set by the person who reports the bug, thus leading to inconsistencies within the bug database [23]. For example, the severity and priority of a bug may change according to person and environment [23]. Also it has been shown that Bugzilla typically uses too many severity levels [29]. Similarly the bug may be reported by a normal user, an expert, or automatically, thus, each entity will report the bug according to their own level of expertise and preference.

Due to these factors previous studies on bug database mining have focused on using the short and long descriptions of the bugs, as they contain the most generalized information about the bug [2], [3].

B. *Bug Reports associated with Linux Vulnerabilities*

As this study investigates the possibility of using bug reports in order to identify software vulnerabilities, it is necessary to discover bug reports that are associated with vulnerabilities. Since some bug reports do not state the specific vulnerability it is associated with, the bug ID associated with each vulnerability in the MITRE CVE [5] database was used to identify these bugs.

TABLE IV. LINUX KERNEL VULNERABILITIES ASSOCIATED WITH BUG REPORTS

Description	Number
Number of hidden impact vulnerabilities with bug reports in Redhat Bugzilla	76
Number of non-hidden impact vulnerabilities with bug reports in Redhat Bugzilla	98
Number of vulnerabilities with bug reports that are not accessible	152
Number of vulnerabilities with bug reports exclusively from Linux Kernel Bugzilla	5
Number of vulnerabilities with no bug reports associated with them	72
Total	**403**

The vulnerability database contained vulnerabilities for the Linux kernel from January 2006 to April 2011. By applying the rules stated in Section III.A, the number of vulnerabilities was reduced to 403. However, 72 vulnerabilities did not have associated bug reports. As mentioned above, most of the remaining vulnerabilities were associated with bug reports from the Redhat Bugzilla [6] database. Only 15 vulnerabilities were associated with bug reports from the Linux Kernel Bugzilla database [28]. Out of these 15 bugs only 5 were associated exclusively with bug reports from Linux Kernel Bugzilla database [28]. Therefore out of the original 388 MITRE CVE listed vulnerabilities, 326 were associated with bugs from the Redhat Bugzilla database [6].

From the remaining 326 vulnerabilities, 197 were non hidden impact vulnerabilities while 129 were hidden impact vulnerabilities.

Although all the remaining 326 vulnerabilities had a bug ID associated with them, 152 of the bug IDs either did not match a bug ID in the Redhat Bugzilla [6] database or the bug reports were not accessible. Therefore, only 76 hidden impact vulnerabilities and 98 non-hidden impact vulnerabilities were matched with a bug report (see Table IV).

V. CLASSIFICATION FOR VULNERABILITY IDENTIFICATION VIA BUG DATABASES

Bug reports in publicly available bug databases are extremely varied due to the fact that the bug reporting systems are not standardized and the expertise and requirements of the bug reporters vary. However, previous work on bug triaging and classification successfully makes use of the short and long descriptions of bug reports.

Thus, we propose a bug classification methodology that

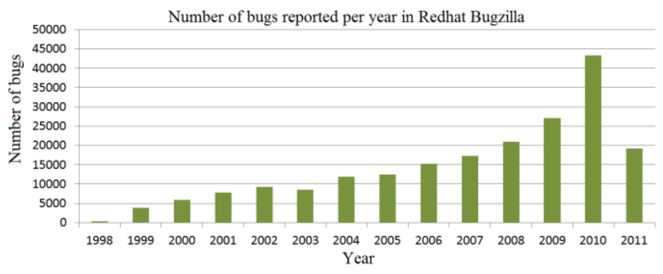

Fig. 6. Number of bugs reported in the Redhat Bugzilla database [6]

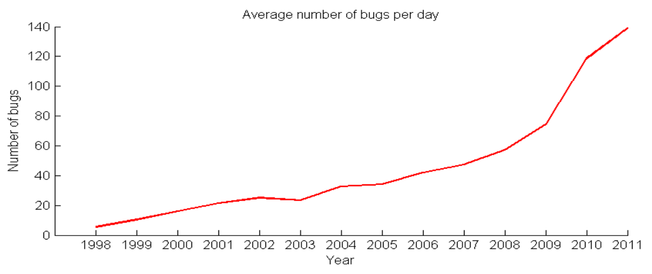

Fig. 7. Average number of bugs per day reported in the Redhat Bugzilla database [6]

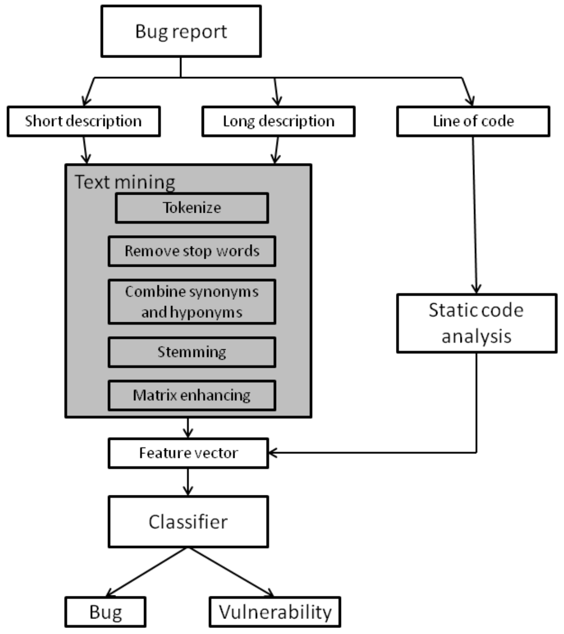

Fig. 8.The proposed vulnerability discovery methodology

uses the short and long descriptions of bug reports in publicly available bug databases, and advanced text mining techniques coupled with machine learning algorithms to aid discovery of hidden impact vulnerabilities. The proposed methodology, illustrated in Fig. 8, will extract the short and long descriptions from a reported bug and generate a feature vector via text mining. The text mining will involve: tokenizing, removal of stop-words, combining synonyms and hyponyms, stemming and matrix enhancing. Furthermore a static code analysis of the location or module the bug was reported in will add to the feature vector. The classifier will then classify the reported bug as a normal bug or a vulnerability using the feature vector.

The implementation of the proposed classifier was initiated for the Linux kernel vulnerabilities by using the Redhat Bugzilla bug database as the source of bug descriptions. The bug reports in the database were divided into three classes: normal bugs (~200,000), bugs that are vulnerabilities (98), and bugs that are hidden impact vulnerabilities (76). The set of vulnerabilities considered were vulnerabilities reported from 2006 to 2011, and the earliest bug report that was associated with a vulnerability was from 2004. Therefore, the set of normal bugs considered for the classifier contained bugs reported from 2004 to 2011. Thus according to Table III, the normal set contained 167,390 bugs.

This general classifier faces two main problems: first the large dimensionality of the feature vector and second the base-rate fallacy problem. The following Sub-Sections illustrates these two problems.

A. Generation of the feature vector

Because of the large number of unique words contained in the textual descriptions of bugs and the large number of bugs considered, the dimensionality of the feature vector will be large. Such a high dimensionality will increase the training time of the classifier as well as the memory and processor requirement. Thus, in order to reduce the dimensionality of the feature vector without losing the most significant information in the bug report, the following methodology was used to extract the feature vector.

First, the long and short descriptions of the bugs were extracted. Tokenization was used to extract the unique words in the descriptions. In the tokenization process special characters and numbers were removed, and capitalization and other text formatting was also removed from the text. Because of the large number of normal bugs, a random sample of 4000 bugs were used for the extraction of unique words. Table V shows the number of unique words in each category of bug.

Second, stop words were removed from the extracted unique word lists. Stop words are words that are commonly used in the English language and do not carry any information. By removing stop words, the number of unique words was reduced without loss to the information contained in the text.

Third, Wordnet [30] was used to identify synonyms and hyponyms and combine these. Synonyms and hyponyms are words that carry the same information in a different form. This step combines words that carry similar information and further reduced the number of unique words.

Porter stemming [31] was performed as the fourth step. In this step words are stemmed into their most basic form. Similar to combining synonyms and hyponyms this step combined words that carry similar information, and further reduced the number of unique words.

As the final step for generating the term document matrix the unique words that occur in less than 10% of the records in each category were removed. This step removes words that are less generalized and reduced the number of unique words further.

The reduction of the size of the feature vector after each step of the text mining process can be seen in Table V. The resulting feature vector contains words that are most generalized to the bug database. Due to the small ratio between the number of bugs that are hidden impact vulnerabilities and the number of normal bugs, textual information from each category was extracted separately and finally combined to obtain the feature vector.

As illustrated in Table V, the initial number of unique words is very large, thus, making the feature vector too large. However, by utilizing Wordnet and Porter stemming the number of unique words were reduced, and after removing the keywords that occur in less than 10% of the bugs, the length of the feature vector was reduced to 633.

By utilizing the feature vector, the term-document matrix is generated. However, this matrix is extremely sparse. Thus, for this application, techniques such as TF-IDF for matrix enhancing will be used to improve the term-document matrix.

B. The Base-Rate fallacy problem

In [32] Axelsson performed a base-rate fallacy test for intrusion detection systems (IDS) and illustrated the problems in classifying intrusions. Axelsson pointed out the small ratio between the number of intrusions and normal traffic affect the outcome in such a way that the user will be overwhelmed by the number of false positives. Since the ratio between hidden impact vulnerabilities and normal bugs in bug databases is very low ($129/167,390 = 7.71 \text{ X } 10^{-4}$), a similar base-rate fallacy evaluation was performed. However, it has to be noted that the number of hidden impact vulnerabilities used for this

TABLE V. NUMBER OF KEYWORDS IN EACH CLASS OF THE DATASET

Type of bug		Total number of unique words after tokenization	After removing stop words	After combining synonyms and hyponyms	After Porter stemming	After removing words that occur in less than 10% of bugs
Non hidden impact vulnerabilities	Short description	335	308	272	268	8
	Long description	2595	2468	1848	1794	159
Hidden impact vulnerabilities	Short description	325	297	264	260	8
	Long description	2144	2026	1564	1525	210
Normal bugs	Short description	6161	6039	4536	4349	90
	Long description	9981	9843	8067	7825	158
Total		**21541**	**20981**	**16551**	**16021**	**633**

calculation is a conservative estimate since 1) although all the bugs reported for 2004 and 2005 were included in the normal bug set, hidden impact vulnerabilities discovered for that time period were not included, and 2) the normal bug set may include bugs that will eventually be discovered as vulnerabilities in the future.

For the base-rate fallacy analysis, the following nomenclature will be used:

$V = hidden\ impact\ vulnerability$

$D = detection\ (i.e.\ the\ classifier\ detects\ a\ bug\ as\ a\ vulnerability)$

$\neg X = not\ X$

$P(X) = probability\ of\ X$

$P(X \mid Y) = probability\ of\ X\ given\ Y$

Thus, by using the above naming convention, true positive rate can be denoted as $P(D \mid V)$ and the false positive rate can be denoted as $P(D \mid \neg V)$.

For classification of vulnerabilities the Bayesian detection rate is the probability that a bug is a vulnerability given that the classifier detects the bug as a vulnerability, i.e. $P(V \mid D)$. In order to increase the Bayesian detection rate the number of false positives must be reduced. By means of Bayes' theorem the Bayesian detection rate can be expressed as:

$$P(V \mid D) = \frac{P(V) \bullet P(D \mid V)}{P(V) \bullet P(D \mid V) + P(\neg V) \bullet P(D \mid \neg V)} \quad (1)$$

The following probabilities are known:

$$P(V) = \frac{129}{167390} = 7.71 \times 10^{-4} \quad (2)$$

$$P(\neg V) = 1 - P(V) = 1 - 7.71 \times 10^{-4} = 0.99923 \quad (3)$$

By using equations (2) and (3), equation (1) can be rewritten as:

$$P(V \mid D) = \frac{7.71 \times 10^{-4} \bullet P(D \mid V)}{7.71 \times 10^{-4} \bullet P(D \mid V) + 0.99923 \bullet P(D \mid \neg V)} \quad (4)$$

The Bayesian detection rate expressed in equation (4) is dominated by the factor 0.99954, i.e. the high probability that a bug is not a vulnerability. Thus in order to achieve a Bayesian detection rate that is sufficient, the false positive rate must be low. Fig. 9 plots the false positive rate against the Bayesian detection rate, for different values of true positive rates ($P(D \mid V)$). Fig. 9 shows that as the false positive rate increases, the Bayesian detection rate decreases.

The Bayesian detection rate is vital when dealing with human users: if the Bayesian detection rate is too low, the users will be overwhelmed by the number of false positives and thus reducing the effectiveness of the classifier. It is not possible to guess what the sufficient level of Bayesian detection rate will be for the classifier. However, by using the upper bound in Fig. 9. it is possible to gain an understanding of the maximum false positive rate which is acceptable from the classifier. For example, if a Bayesian detection rate of 0.01 can be tolerated by the development team, which means that only one out of 100 detections is an actual vulnerability, according to Fig. 9, a maximum false positive rate of 0.076 is acceptable. This means that on average for any given day in 2011, where 139 bugs were reported per day (see Table III), around 11 (0.076 * 139) bugs will be falsely identified as a vulnerability by the classifier. Similarly, if one out of 10 detections needs to be an actual vulnerability, which means a Bayesian detection rate of 0.1, to achieve this, the maximum acceptable false positive rate is 0.0069. This translates to falsely identifying around one bug per day (0.0069 * 139) for any given day in 2011. Thus, the lower boundary of false positive rate that the proposed classifier must obtain can be determined using Fig. 9.

VI. CONCLUSION

More effective vulnerability discovery and identification is an important factor in the software life cycle as it will reduce the security exposure of vital systems. The earlier a reported bug is identified as a vulnerability the more effective developers can be in identifying which bugs have higher priority for patch creation (vulnerabilities have a high priority), and the more effective system owners can be in choosing which patches should be applied quickly.

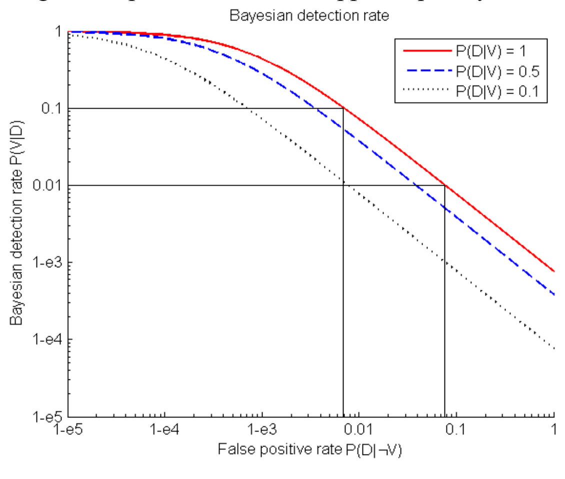

Fig. 9. Bayesian detection rate for classifying vulnerabilities.

An analysis of the most significant Linux kernel vulnerabilities and MySQL vulnerabilities showed a significant number of vulnerabilities that affected the Linux kernel (39.4%) and the MySQL database server (62.23%) were hidden impact vulnerabilities and it was shown that the percentage of hidden impact vulnerabilities has increased in the last two years. Thus there is a necessity to use bug databases to identify hidden impact vulnerabilities in software.

A further analysis of the Redhat Bugzilla Linux bug database showed the difficulties of data mining such databases. However an analysis of previous research into bug triaging and classification showed that the information contained in bug reports can be used for classification purposes. This paper also proposed a system that utilizes bug reports to identify hidden impact vulnerabilities. Potential problems faced by a development team when using the proposed classifier were also addressed in the paper.

As future work the proposed system will be implemented for discovering vulnerabilities in the Linux kernel, MySQL and other third party software. The system will use advanced text mining techniques and machine learning algorithms to classify bugs and vulnerabilities. In order to further enhance the classification accuracy, attributes of the source code itself and other aspects of the software development process will be incorporated into the classifier.

REFERENCES

[1] 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.

[2] 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)*, May 2010, pp.1–10.

[3] 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)*, Mar. 2011, pp.249–258 .

[4] 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)*, Sep. 2006, pp. 127–134.

[5] The MITRE Corporation (1 Nov. 2011), *Common Vulnerabilities and Exposures (CVE)* [Online]. Available: http://cve.mitre.org/.

[6] Redhat, Inc. (1 Nov. 2011), *Redhat Bugzilla Main Page* [Online]. Available: https://bugzilla.redhat.com/.

[7] F. Yamaguchi, F. 'FX' 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.

[8] 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)*, Sep. 2011, pp. 207–214.

[9] P. Li, B. Cui, “A comparative study on software vulnerability static analysis techniques and tools,” in *Proc. of the IEEE Int. Conf. on Information Theory and Information Security (ICITIS)*, Dec. 2010, pp.521–524.

[10] 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)*, Sep. 2011, pp. 97–106.

[11] D. Kester, M. Mwebesa, J. S. Bradbury, “How Good is Static Analysis at Finding Concurrency Bugs?” in *Proc of the 10th IEEE Int. Working Conf. on Source Code Analysis and Manipulation (SCAM ’10)*, Sep. 2010, pp. 115–124.

[12] W. M. Khoo, S. Aloteibi, R. Anderson, M. Meeks, “Hunting for vulnerabilities in large software: the OpenOffice suite,” *Cambridge University press*, Jun. 2010.

[13] M. Schumacher, C. Haul, M. Hurler, A. Buchmann, “Data Mining in Vulnerability Databases,” in *Proc of 7th Workshop "Sicherheit in vernetzten Systemen"*, Mar. 2000.

[14] L. Torri, G. Fachini, L. Steinfeld, V. Camara, L. Carro, É. Cota, “An Evaluation of Free/Open Source Static Analysis Tools Applied to Embedded Software,” in *Proc of the 11th Latin American Test Workshop (LATW ’10)*, Mar. 2010, pp. 1–6.

[15] K. Kratkiewicz, R. Lippmann, “Using a Diagnostic Corpus of C Programs to Evaluate Buffer Overflow Detection by Static Analysis Tools,” in *Proc of Workshop on the Evaluation of Software Defect Detection Tools*, Jun. 2005, pp. 62–71.

[16] M. Zitser, R. Lippmann, T. Leek “Testing Static Analysis Tools Using Exploitable Buffer Overflows From Open Source Code,” in *Proc. of the 12th Int. Symp. on Foundations of Software Engineering (FSE ’04)*, ACM SIGSOFT, Nov. 2004, pp. 97–106.

[17] D. Cubranic, G. C. Murphy, “Automatic bug triage using text categorization,” in *Proc. of the 16th Int. Conf. on Software Engineering and Knowledge Engineering*, KSI Press, Jun. 2004, pp. 92-97.

[18] J. Anvik, L. Hiew, G. C. Murphy, “Who Should Fix This Bug?” in *Proc. of the 28th Int. Conf. on Software Engineering (ICSE ’06)*, May 2006, pp. 361–370.

[19] G. Jeong, S. Kim, T. Zimmermann, “Improving bug triage with bug tossing graphs,” in *Proc. of the 7th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT (ESEC/FSE ’09)*, Aug. 2009, pp. 111–120.

[20] P. Runeson, M. Alexandersson, O. Nyholm, O, “Detection of Duplicate Defect Reports Using Natural Language Processing,” in *Proc. of the 29th Int. Conf. on Software Engineering (ICSE 2007)*, 20-26 May 2007, pp.499–510.

[21] X. Wang, L. Zhang, T. Xie, J. Anvik, J. Sun, “An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution Information,” in *Proc. of the 30th Int. Conf. on Software Engineering (ICSE '08)*, May 2008, pp. 461–470.

[22] 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)*, Sep. 2011, pp. 8:1–8:9.

[23] L. Wu, B. Xie, G. Kaiser, R. Passonneau, “BugMiner: Software Reliability Analysis Via Data Mining of Bug Reports,” in *Proc. of the 23rd Int. Conf. on Software Engineering and Knowledge Engineering (SEKE)*, Jul. 2011, pp. 95–100.

[24] H. Shahriar, M. Zulkernine, “Classification of Static Analysis-based Buffer Overflow Detectors,” in *Proc of the 4th Int. Conf. on Secure Software Integration and Reliability Improvement Companion*, Jun. 2010, pp. 94–101.

[25] 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)*, Sep. 2011, pp. 30–39.

[26] 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.

[27] M. F. Ahmed, S. S. Gokhale, “Linux Bugs: Life Cycle, Resolution and Architectural Analysis,” in *Information Software Technology*, vol. 5, no. 11, pp. 1618–1627, Nov. 2009.

[28] Linux Kernel Organization (1 Nov. 2011), *The Linux Kernel Archives* [Online]. Available: http://www.kernel.org/.

[29] I. Herraiz, D. M. German, J. M. Gonzalez-Barahona, G. Robles, “Towards a simplification of the bug report form in Eclipse,” in *Proc. of the Int. Working Conf. on Mining Software Repositories (MSR '08)*, May 2008, pp. 145–148.

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

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

[32] 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.

[33] T. Menzies, A. Marcus, “Automated Severity Assessment of Software Defect Reports,” in *Proc of the IEEE Int. Conf. on Software Maintenance*, Sep. 2008, pp. 346-355.