Identifying Security Bug Reports via Text Mining: An Industrial Case Study

[1]Michael Gegick, [2]Pete Rotella, [3]Tao Xie
[1]Independent, [2]Cisco Systems , [3]North Carolina State University Department of Computer Science
mcgegick@gmail.com, protella@cisco.com, xie@csc.ncsu.edu

Abstract -- **A bug-tracking system such as Bugzilla contains bug reports (BRs) collected from various sources such as development teams, testing teams, and end users. When bug reporters submit bug reports to a bug-tracking system, the bug reporters need to label the bug reports as security bug reports (SBRs) or not, to indicate whether the involved bugs are security problems. These SBRs generally deserve higher priority in bug fixing than not-security bug reports (NSBRs). However, in the bug-reporting process, bug reporters often mislabel SBRs as NSBRs partly due to lack of security domain knowledge. This mislabeling could cause serious damage to software-system stakeholders due to the induced delay of identifying and fixing the involved security bugs. To address this important issue, we developed a new approach that applies text mining on natural-language descriptions of BRs to train a statistical model on already manually-labeled BRs to identify SBRs that are manually-mislabeled as NSBRs. Security engineers can use the model to automate the classification of BRs from large bug databases to reduce the time that they spend on searching for SBRs. We evaluated the model's predictions on a large Cisco software system with over ten million source lines of code. Among a sample of BRs that Cisco bug reporters manually labeled as NSBRs in bug reporting, our model successfully classified a high percentage (78%) of the SBRs as verified by Cisco security engineers, and predicted their classification as SBRs with a probability of at least 0.98.**

I. INTRODUCTION

Software organizations use bug-tracking systems (BTSs) such as Bugzilla[1] to manage bug reports (BRs) collected from various sources including development teams, testing teams, and end users. In a BTS, some BRs are labeled by bug reporters as security bug reports (SBRs), whose associated bugs are found to be security problems. SBRs generally deserve higher fix priority than not-security bug reports (NSBRs), the subset of BRs that are believed not to have a security impact.

Correctly labeling SBRs among BRs submitted to a BTS is important in security practice since delay of identifying and fixing the security bugs involved in the SBRs causes serious damage to software-system stakeholders. The likelihood of unlabeled SBRs in a BTS could be high for at least three reasons. First, if bug reporters perceive a subtle security bug that they are reporting in a BR as an innocuous not-security bug, then they may label the BR as an SBR. Second, some security bugs described in BRs are associated with recommended mitigations that may be unknown to bug reporters. For example, if a SQL parser throws an exception due to input containing a single quote, then a bug reporter without sufficient security knowledge may report this bug as a NSBR, whose related bug may be later fixed by filtering the input for single quotes. However, attackers can write crafty exploits to circumvent such filtering [1]. A bug reporter with sufficient security knowledge would realize that single quotes can be used in SQL injection attacks and report this bug as an SBR, whose related bug would be later fixed by limiting privileges on a database server and using prepared statements that bind variables as advised by Howard et al. [7]. Third, a bug related to general reliability problems can also be related to security problems [15] and a bug reporter without sufficient security knowledge may report this bug as a NSBR. For example, a bug that causes a system to crash can also be a denial-of-service security bug if exploited by an attacker.

In the practice of bug reporting, bug reporters may often mislabel SBRs as NSBRs partly due to lack of security domain knowledge as discussed earlier. Then it is desirable for security engineers to inspect NSBRs submitted to a BTS to identify SBRs that are manually-mislabeled as NSBRs in the BTS. However, manually inspecting (often thousands of) NSBRs in a BTS to identify SBRs is time-consuming and often infeasible, or not even conducted in practice. For example, to the best of our knowledge, the open source project members of Mozilla and Red Hat[1,2] do not have the practice of manually inspecting each NSBR submitted to their BTSs, while acknowledging that some NSBRs in their BTSs were in fact mislabeled, and should have been SBRs[3].

Therefore, there remains a strong need of effective tool support for reducing human efforts in this process of identifying SBRs in a BTS, enabling this important security practice of SBR identification in either industrial or open source settings. With such effective tool support, security engineers can elevate the priority of each identified SBR and ensure that the described security bug receives appropriate fortification efforts, and gets fixed timely, thus improving the security assurance of the software.

To satisfy such a strong need, in this paper, we propose a new tool-supported approach that applies text mining on

[1] https://www.mozilla.org/projects/security/security-bugs-policy.html
[2] http://ovasik.fedorapeople.org/bugs.pdf
[3] http://mail.opensolaris.org/pipermail/tools-discuss/2009-March/004379.html

natural-language descriptions of BRs to learn a statistical model to classify[4] a BR as either an SBR or an NSBR. With the help of our approach, security engineers can feasibly apply our proposed approach on NSBRs from a BTS to effectively identify SBRs, without inspecting each single NSBR from a BTS (which is simply infeasible in practice as discussed earlier).

The rationale of our proposed approach is to exploit valuable natural-language information of BRs in a BTS. Although bug reporters may not recognize that the bug they are describing is a security bug, the natural-language description of the bug in the BR may be adequate to indicate that the bug is security-related and thus the BR is an SBR.

To identify SBRs by exploiting valuable natural-language information in BRs, we propose an approach that learns a natural-language statistical model for classifying a BR as either an SBR or NSBR. We implement our approach based on an industrial text mining tool called SAS Text Miner[5]. We evaluated our model on a large Cisco software system that contains over ten million source lines of code (SLOC). We trained our model on four years of Cisco SBRs and then applied the model on BRs that were labeled as NSBRs by Cisco bug reporters. We also applied the model on BRs from three additional large Cisco software systems, two of which each consist of over five million SLOC, and one of which consists of over 10 million SLOC.

In this paper, we use the terms "bug reporters," "software engineers," and "security engineers" as follows. A bug reporter is any person (internal or external to Cisco) who reports a bug to the Cisco BTS. A software engineer is any stakeholder at Cisco responsible for the development of Cisco software. Software engineers may also be bug reporters. A security engineer is a Cisco engineer that is responsible for securing Cisco software. Security engineers assess SBRs that have been submitted to the BTS.

In summary, this paper makes the following main contributions:

- The first approach that learns a natural-language model to automate the classification of SBRs. We also show how our model can be trained and refined to improve the effectiveness of classifying SBRs mislabeled as NSBRs.
- An extensive empirical evaluation of the proposed approach on four large Cisco software systems. Two systems each consist of over five million SLOC and the other two systems each consist of over ten million SLOC. Our results indicate that our model can classify seven times more SBRs from a BTS of one system than randomly selecting BRs from the BTS (a default strategy when the number of BRs is beyond the security engineers' afforded inspection efforts). Among a sample of BRs that Cisco bug reporters originally labeled as NSBRs, our model successfully identified a high percentage (78%) of the SBRs as verified by a Cisco security engineers, and predicted their classification as SBRs with a probability of at least 0.98.

[4] We reserve *labeling* for manually identifying BRs as SBRs or NSBRs and *classifying* for model-based identification of SBRs and NSBRs.

[5] http://www.sas.com/technologies/analytics/datamining/textminer/

Due to the very promising results of our case study, Cisco is planning on carrying out another pilot study, confirming the value and impact of our work on Cisco practice. The rest of this paper is organized as follows. Section II provides background. Sections III and IV detail our approach and case study setup. Section V presents the results. Section VI discusses the threats to validity. Section VII provides related work, and Section VIII concludes.

II. TEXT-MINING OVERVIEW

Text mining uses natural-language processing to parse terms (i.e., words and phrases) from a document to create a term-by-document frequency matrix. Table I shows a simple, hypothetical term-by-document matrix. A document is represented by a vector (column) in the matrix that contains the number of times each of the different terms occurs in the document. The matrix provides a quantitative representation of the document that text mining uses to classify documents. The models that use such matrices to represent documents are called vector space models, and are commonly used for text mining [11].

The investigator performing text mining can decide what terms to enter into the matrix by creating a pre-defined list of terms. A start list contains terms that are most likely to be indicative of categories of documents. If terms in a document match those in the start list, then those terms are entered into the matrix.

Table I. A term-by-document frequency matrix.

Term	Document 1	Document 2	Document 3
Attack	1	0	1
Vulnerability	1	0	0
Buffer overflow	3	0	0

A stop list contains terms such as articles, prepositions, and conjunctions that are not used in text mining. If terms in the stop list match those in a document, then those terms are not entered into the matrix. While the terms in the start list are not unique to SBRs, their frequency and presence with other security terms in a BR increase the probability that the BR is an SBR. The synonym list contains terms with the same meanings (e.g., "buffer overflow" and "buffer overrun" have the same meaning). Terms in a synonym list are treated equivalently in text mining. Therefore, a less-used term that is associated with SBRs may be given more weight in the predictive model if the term is synonymous with a term that is often used with SBRs.

Weighting functions can be assigned to the terms and their frequencies in each vector. The total weight of a term is determined by the frequency weight and the term weight. We use the log frequency weight function to lessen the effect of a single term being repeated often in each BR. We use the entropy term weight function to apply higher weights to terms that occur infrequently in the BRs [14]. A statistical model can then estimate the probability that a document belongs in a given category based on the weighted values in the vector. Being a major part of text mining, text classification uses a built natural-language predictive model to classify documents into predefined categories with a pre-classified training set [14].

III. APPROACH

Our approach consists of three main steps. The first step is to obtain a labeled BR data set that contains textual descriptions of bugs and labels to indicate whether a BR is an SBR or an NSBR. The labeled BR data set is required for building and evaluating our natural-language predictive model. The second step is to create three configuration files that are used in text mining: a start list, a stop list, and a synonym list. The third step is to train, validate, and test the predictive model that estimates the probability that a BR is an SBR.

A. *Textual-Data Preparation*

The textual-data step prepares labeled data for building and evaluating our natural-language predictive model. This step includes three sub-steps. First, from a BTS, we obtain BRs that were submitted by stakeholders including development team, testing teams, and end users. Second, we distinguish between SBRs and NSBRs among the obtained BRs. In some commercial software organizations, a BR contains a label field that indicates whether the BR is an SBR. A query on this field in the BTS causes all known SBRs to be returned. If the field is not present in the BTS or an insufficient number of BRs are labeled, then manual efforts from software or security engineers are needed to label a subset of all BRs as SBRs or NSBRs. In the end, we use labeled BRs to build and evaluate our natural-language predictive model. Finally, using a built-in function in SAS, we enumerate all the terms in the labeled SBRs and NSBRs. These terms are necessary for the next step where we construct configuration files. Generally, the more labeled data used for building the predictive model, the more accurate the predictive model is. According to SAS [14], the minimum count of documents required for natural-language modeling is 100.

B. *Configuration-File Preparation*

After we obtain the terms from the BRs, we select terms from them to prepare the start, stop, and synonym lists. To the start list, we manually add terms such as "vulnerability" and "attack" from SBRs. We also include terms (from SBRs) that are not explicitly security-related, but can indicate a security problem. For example, "crash" and "excessive" are also candidates for inclusion in the start list.

To the stop list, we add prepositions, articles, and conjunctions since they likely have little benefit for indicating a security bug. Both stop lists and start lists are acceptable for text mining [14]. SAS Text Miner allows either a start list or stop list to be used in text mining, but not both. In our approach and case study, we tried each type, and experienced similar classification results.

To the synonym list, we add synonyms based on examinations of the enumerated terms from SBRs and NSBRs. Bug reporters may use security-related verbiage such as "buffer overflow" or "buffer overrun" to describe the same bug. By including such terms in the synonym list, the predictive model can identify different terms in the same context to reflect the same type of bugs.

We next use SAS Text Miner to generate a term-by-document frequency matrix from the terms in BRs based on the start or stop, and synonym lists. The matrix is a quantified format of the natural language descriptions in the BRs. If we include a large number of BRs in text mining, the term-by-document frequency matrix can become large. A large matrix can hinder the predictive modeling in text mining [5]. We reduce the size of the matrix by choosing the singular value decomposition (SVD) option in SAS. SVD determines the best least squares fit to the weighted frequency matrix, based on a preset number of terms, k [14]. High (30-200) values of k are useful for prediction whereas small (2 to 50) values of k are more effective for clustering similar documents [14]. We use 200 for the value of k for our text classification, which is for prediction instead of clustering.

C. *Predictive Modeling*

Next, we use the term-by-document matrix as the independent variable (i.e., the input variable) in our predictive model. The dependent variable (i.e., the value that we intend to predict) is the label (SBR or NSBR) of a BR. We apply SAS Text Miner to construct a trained model based on the term-by-document matrix. The recall and precision of the trained model enable us to judge whether we need to reassess the content of the configuration files or the value of k for SVD. If the results are satisfactory, then the trained model is usable and we can feed a new BR data set (e.g., BRs without labels) to the model for predicting their labels. We next describe the training, validation, and test data sets used for training the model, the application of our trained model on new BRs, and retraining of the model with corrected mislabeling (when the initial training data include mislabeled data).

i. Training, Validation, and Test Data Sets

First, we train, validate, and test the model using the BR data set that we earlier prepared, and divide the BR data set into three smaller data sets: the training, validation, and test data sets [13]. The training data set is used for preliminary model training. The validation data set is used for selecting the optimum configuration options (such as weights for the term vector in the matrix). The test data set is used for an assessment of the model for the data that have not been used to train or validate the model. The proportions of BRs allocated to the training, validation, and test data sets are 60%, 20%, and 20%, respectively, as recommended by SAS [14].

ii. Application of Trained Model on new BRs

Given new BRs (e.g., BRs without labels), our built predictive model then estimates the probability that a BR is an SBR. In our setting, the probability ranking is a list of BRs sorted in descending order of the estimated probability of being an SBR. Security engineers can start their assessments of the BRs at the top of the probability ranking and continue until they reach a pre-defined probability threshold. The threshold indicates that SBRs with probabilities below the threshold may exist, but there are only few of them.

We determine the probability threshold with the following technique. We first assess the probabilities of SBRs. If all the SBRs have higher probabilities than the NSBRs, we assign the threshold as the lowest probability associated with an SBR. If some NSBRs have higher probabilities than some SBRs, the threshold must be made based on the security engineers' available resources. In particular, based on the results from the test set, we can determine the lowest estimated probabilities assigned to SBRs. Security engineers should look at the lowest probability of an SBR in the test set and then match that probability to the probability ranking of the BRs that they intend to assess. If there are too many BRs above that probability to assess, then security engineers should use the next lower threshold, and so on.

iii. Model Retraining with Corrected Mislabeling

One inherent challenge in our research context is the (un)certainty of the labeling of SBRs (by bug reporters) initially used for training the model. The first author's empirical investigations (with security engineers in software organizations other than Cisco) have revealed that SBRs are sometimes mislabeled as NSBRs by bug reporters. If we train the model on SBRs mislabeled as NSBRs, the model may classify security-related verbiage as not security-related, and, as a consequence, incorrectly classify an SBR with security-related verbiage as an NSBR. Therefore, the model's accuracy would likely be improved if security engineers review each BR used to train the model to ensure that the BR's label is correct. To address this issue, we select a subset of the NSBRs from the BTS. Then, we submit the NSBRs to security engineers for them to check for mislabeling. If any true SBRs exist among the labeled NSBRs (which are thus mislabeled), then we retrain the model with the subset of NSBRs that are now correctly labeled. The verbiage between SBRs that contain explicit security verbiage (e.g., attack) may be distinctly different than SBRs that are mislabeled as NSBRs and do not contain such explicit security verbiage. By training the model on SBRs that are mislabeled as NSBRs (in addition to true NSBRs), the model can classify SBRs with terms that bug reporters are likely to use to describe security bugs when they do not realize the problem to be security-related. Additionally, from the original configuration files, we add or subtract terms from the original configuration-files that appear in the SBRs and NSBRs that were reviewed by the security engineers.

IV. CASE STUDY SETUP

We next describe the four large Cisco systems under study, the research questions that we intend to answer using studies of these systems, and our study design to address these questions.

A. Four Cisco Software Systems

We analyzed four large Cisco software systems, referred to as Systems A, B, C, and D. Identities of these systems cannot be disclosed here due to confidentiality. Each system is implemented primarily in the C programming language. Systems A and B consist of over ten million SLOC each and Systems C and D consist of over five million SLOC each. Cisco's BTS contains all BRs associated with these software systems, and these BRs document both bugs and failures in the software systems. Each BR contains a field that is manually filled (initially by bug reporters) to label the BR as an SBR. Security engineers can then evaluate a labeled SBR to either verify that the BR is in fact an SBR, or, if not, reset the field to indicate an NSBR. Each BR also contains a summary text field and a larger description text field. Our text mining focuses on these two text fields of BRs that have a severity rating of 1, 2, or 3, out of the range of 1-6 where severity 1 has the most detrimental impact on the system. The severity ratings were assigned by Cisco software engineers.

B. Research Questions

In our studies, we address the following research questions:

- RQ1: How effective is our model at classifying SBRs of a given system if the model is trained on a BR data set from the *same* system?
- RQ2: Do bug reporters fail to recognize that some BRs are SBRs?
- RQ3: How effective is our model at classifying SBRs that are manually-mislabeled as NSBRs? How much

negative impact would training the model on SBRS manually-mislabeled as NSBRs cause on applying our approach?

- RQ4: How effective is our model at classifying unlabeled SBRs in a given system if the model is trained on a BR data set from a *different* system?

The answer to RQ1 helps us to assess the effectiveness of our approach when applied to cases where the bug reporter describes a security problem or a non-security problem in the BR description, but does not label the BR (i.e., not labeling the BR as an SBR or a NSBR).

The answer to RQ2 helps us to determine whether Cisco security engineers should review the Cisco BTS for SBRs that are manually-mislabeled as NSBRs by bug reporters.

The answer to RQ3 helps us to determine whether our approach is effective in automatically classifying SBRs that bug reporters manually-mislabel as NSBRs. If such BRs exist, then we should include the terms associated with these SBRs in our model, since they describe real security problems, but do not explicitly use security-related verbiage (e.g., attack).

The answer to RQ4 helps us to assess the effectiveness of our approach in classifying SBRs in other systems that the model was not trained on. The results can indicate whether the bug reporters or security engineers can obtain assistance from our approach in automatically labeling unlabelled BRs of other systems. Using a common model across systems would reduce training and modeling efforts and result in providing additional training data across systems for the model.

C. *RQ1 Study Setup*

We first queried the Cisco BTS for manually-labeled SBRs associated with System A for the past four years. Next, we randomly sampled System A's BRs that were manually-labeled as NSBRs by bug reporters. The samples of SBRs and NSBRs are equal in counts to provide the model with enough data to classify both SBRs and NSBRs accurately. We call the data set of SBRs and NSBRs for System A the $\mathbf{A}_{\mathbf{feasibility}}$ data set. We randomly partition $A_{feasibility}$ into training, validation, and test data sets. We call the model that is trained on the $A_{feasibility}$ data set the "**trained**" model. The initial results are used to calibrate the model and provide an assessment of the predictive power of BR descriptions. The results would not indicate whether the model correctly classifies an SBR that was mislabeled as an NSBR.

D. *RQ2 Study Setup*

We randomly sampled BRs (from System A) that were manually-labeled as NSBRs by bug reporters. We call this data set $\mathbf{A}_{\mathbf{pilot}}$. We applied the trained model on A_{pilot} to estimate the probability that a manually-labeled NSBR is an SBR. We then submitted A_{pilot} to the security engineers for them to review the same content (that the model used for prediction) to determine whether any of the manually-labeled NSBRs are actually SBRs. We did not reveal the estimated probabilities to the security engineers to reduce potential bias in their analyses. Based on prior discussions with the security engineers, we estimated that security engineers would require approximately 175 person-hours to analyze A_{pilot} and determine whether the manually-labeled NSBRs are actually SBRs. At least two security engineers independently reviewed each BR. If two security engineers disagreed on their evaluations of a manually-labeled BR, then they discussed their differences and reached an agreeable consensus. We compared their evaluations with the model's estimated probabilities to evaluate the model's predictions.

E. *RQ3 Study Setup*

Having the security engineers evaluate each BR in A_{pilot} enables us to be certain of the label of each BR in A_{pilot}. We then retrain the model on A_{pilot} (which does not include any BRs from $A_{feasibility}$) to determine whether a model trained on SBRs mislabeled as NSBRs can be useful for the specific purpose of classifying those SBRs that are manually-mislabeled as NSBRs. We allocate A_{pilot} in the 60% (training), 20% (validation), 20% (test) proportions as with the trained model. We refine the start list and synonym list from $A_{feasibility}$, based on the evaluations by the security engineers to focus on terms that bug reporters use when they describe an SBR and do not realize that the bug is a security bug. We call the model that is trained on A_{pilot} the "**retrained**" model.

F. *RQ4 Study Setup*

The security bugs associated with System A may be specific for that software system. Additionally, the bug reporters for System A may have different writing styles and diction for describing bugs than bug reporters from other software systems. To investigate these possibilities, we randomly sampled six months of bug reports from Systems B, C, and D and combined them into one data set that we call **BCD**. We tested whether the trained model, constructed using data from System A, can effectively classify SBRs for three different Cisco software systems. If the model that is trained on System A is predictive for Systems B, C, and D, then the model may be applicable for many other Cisco software systems. Table II provides a summary of our two models, and the three data sets used to train, validate, and test the models. The "Train" column represents the data set that was used to train the model, the "Validate" column represents the data set used to validate the model, and the "Test" column represents the data set used for the model's evaluation.

Table II. Summary of models and data sets.

	Data sets		
Model name	Train	Validate	Test
Trained	$A_{feasibility}$	$A_{feasibility}$	$A_{feasibility}$
			A_{pilot}
			BCD
Retrained	A_{pilot}	A_{pilot}	A_{pilot}

V. RESULTS

We found that a model used with $A_{feasibility}$ with a start list and synonym list identified approximately the same count of manually-labeled SBRs as a model with a stop list. We chose to use a start list and synonym list for our text mining because we suspect that continually updating the start list is more feasible for a limited number of security bugs than managing a large stop list.

If the model classifies an SBR as an NSBR, or if the model classifies an NSBR as an SBR, then the result is a misclassification. We now define the correct classifications and misclassifications for the natural-language model. A true positive (TP) is a verified (by a security engineer) SBR that is correctly classified by the model. A false positive (FP) is a verified NSBR that is incorrectly classified to be an SBR. A false negative (FN) is a verified SBR that is incorrectly classified to be an NSBR. A true negative (TN) is a verified NSBR that is correctly classified to be an NSBR. The success rate of the model is the number of correct classifications divided by the total number of classifications [17]. Model precision is the percentage of correctly classified SBRs among SBRs and NSBRs that have been classified by the model to be SBRs (i.e., exceeding a minimum probability). In our setting, recall is the percentage of correctly classified SBRs (above a minimum probability) among all verified SBRs. The formulas for the success rate, precision, and recall are provided below.

$$\text{Success rate} = \frac{\text{TP + TN}}{\text{TP + FP + TN + FN}} \text{ x } 100\%$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP + FP}} \text{ x } 100\%$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP + FN}} \text{ x } 100\%$$

In the rest of this paper, an SBR denotes a verified SBR unless otherwise stated. The SBR is either verified by the Cisco security engineers before the case study began or is verified as an SBR by the security engineers during our study.

A. Lift Curves and Tables

We measure the effectiveness of the model with lift curves [17], which quantify how much the model improves the rates of classifying SBRs, compared to randomly selecting and analyzing BRs from the BTS. Figure I shows the lift curves for the trained model tested on $A_{feasibility,}$ A_{pilot}, and BCD, and the retrained model tested on A_{pilot}. We explain the details of these results in later subsections. The lift curve x-axis represents the BRs sorted in descending order of likelihood of being an SBR, as predicted by the model, and then divided into ten deciles, where the leftmost decile contains the BRs with the highest likelihood of being an SBR. The y-axis represents the percentage of total SBRs (called the "cumulative" percentage) contained in a given decile classified by the model. The lift curves are cumulative in the sense that the counts of SBRs and BRs are aggregated within each of the ten deciles. An accurate model is one in which the highest SBR rate occurs in the first decile, the second highest in the second decile, and so on. The horizontal dashed lines in Figure I are baselines that represent the rate of classifying SBRs for the data set's deciles if we manually select BRs from the BTS. The overall SBR rate is equal to the count of SBRs divided by the count of all BRs. For a given decile, the difference between the cumulative SBR rate that is derived from the model and the baseline rate represents the effectiveness of the model's classification for that decile. The specific values on the y-axis in Figure I are not disclosed in order to conceal the Cisco SBR rate.

In Table III, we show the cumulative SBR lift values for each decile for each data set shown in Figure I. The cumulative lift value in the first decile is equal to the SBR rate of the model in the first decile divided by the overall SBR rate [13]. The cumulative lift value in the second decile is equal to the cumulative SBR rate of the model divided by the overall SBR rate for the second decile, and so on. The rest of this section compares the model's predictions to randomly selecting BRs from the BTS.

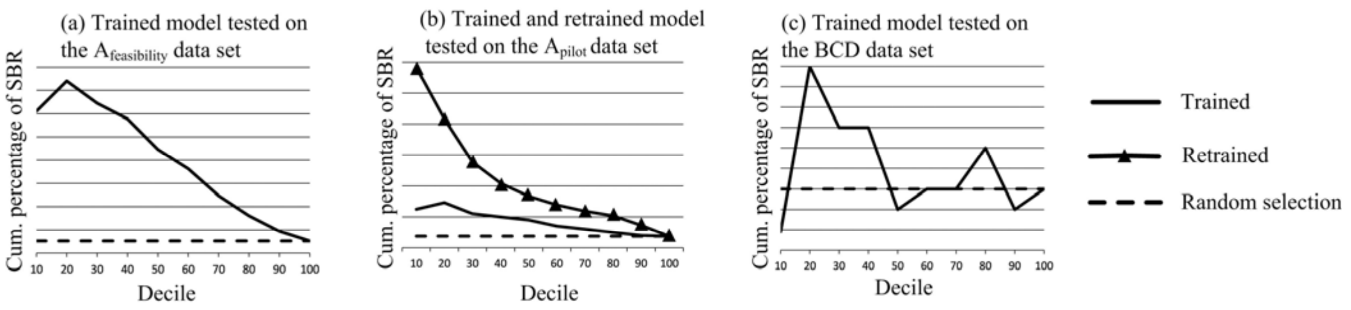

Figure I. Lift curves for case study results.

i. Classifying SBRs in System A

RQ1: How effective is our model at classifying SBRs of a given system if the model is trained on a BR data set from the same system?

The lift curve for the model that is trained, validated, and tested on $A_{feasibility}$ (see Figure Ia) indicates that the chance of finding an SBR generally decreases as the model's estimated probabilities decrease. Although the first decile (associated with the highest probabilities) contains some SBRs, the highest percentage of SBRs exists in the second decile. The increase ("lift") for the first decile is only 1.53, and is 1.65 in the second decile, as shown in Table III for $A_{feasibility}$. The results indicate that security engineers would classify 1.53 times more SBRs in the first decile with the model than by randomly selecting BRs from the BTS. Therefore, this result shows that the verbiage in SBR text fields can be successfully used to classify other unlabeled SBRs. While this lift is low, the lift is nevertheless positive, providing enough justification to continue our analyses on SBRs mislabeled as NSBRs.

Table III. Cumulative lift values for three data sets.

	Data sets			
Decile	$A_{feasibility}$	A_{pilot}	A_{pilot} (retrained)	BCD
1	1.53	3.33	7.00	0.09
2	1.65	3.89	5.00	1.32
3	1.56	2.96	3.33	1.18
4	1.50	2.22	2.50	1.16
5	1.37	1.78	2.00	1.05
6	1.30	1.67	1.67	0.96
7	1.19	1.43	1.43	1.00
8	1.10	1.25	1.25	1.00
9	1.04	1.11	1.11	0.96
10	1.00	1.00	1.00	1.00

Our natural-language model has moderate success in classifying SBRs that bug reporters realize as true SBRs.

ii.Identification of SBRs Mislabeled as NSBRs

RQ2: Do bug reporters fail to recognize that some BRs are SBRs?

The security engineers verified that some[†] of the BRs that were labeled as NSBRs by bug reporters are actually SBRs. Figure Ib shows the lift curve when the trained model (i.e., trained on $A_{feasibility}$) is used to classify SBRs in A_{pilot}. The cumulative lift value for the first decile is 3.33 (see Table III), indicating that security engineers would classify 3.33 times more SBRs that are mislabeled as NSBRs by using the model than they would by randomly selecting BRs from the BTS and verifying the selected BRs.

The model identified several[†] types of security bugs demonstrating that the model does not classify only one type of security bug. One of the SBRs identified in the analysis was also reported in the field, thereby indicating that the model can be used to classify BRs that are found both internally and externally to Cisco. If the SBRs discovered by the security engineers had not already been fixed, it would have received either an elevated priority or would have subjected to a careful security review.

The lift curve for the retrained model (i.e., retrained on A_{pilot}) shows a consistent decrease in lift from the first decile to the tenth decile (see Figure Ib). This result indicates that as the estimated probabilities decrease, the likelihood of being an SBR also decreases. The largest cumulative lift value in our case study, 7.00, is in the first decile for the retrained model. The language used to describe the SBRs in $A_{feasibility}$ may closely resemble the SBRs mislabeled as NSBRs in A_{pilot}; this factor is likely to be responsible for improving the accuracy of the retrained model. Similarly, the NSBRs in A_{pilot} may have resemblance to the NSBRs in $A_{feasibility}$. For each run of the model, security engineers can add examined BRs to the training and validation sets to improve the model's accuracy. Additionally, security engineers can add or

[†]The counts, percentages, and types of security bugs are confidential.

subtract terms (collected from SBRs identified by the security engineers) to the start and synonym lists.

As mentioned earlier, A_{pilot} is the only data set in our study in which security engineers reviewed each BR in the data set. We are therefore certain which BRs are SBRs and which are NSBRs. This certainty improves the retrained model's results over the trained model's results for two reasons. First, the certainty can improve the model training and validation compared to other training and validation data sets that may have SBRs mislabeled as NSBRs. Training the model with SBRs mislabeled as NSBRs can result in the model's misclassification of SBRs as NSBRs. Second, the certainty improves the accuracy of the model in the test data set in A_{pilot} compared to evaluations in other test data sets. If an SBR is mislabeled as an NSBR by a bug reporter in $A_{feasibility}$, but the model classifies the BR as an SBR, then the result is a false positive. The cumulative lift values decrease due to instances where the model is correct, but the labeling of the BRs is incorrect. The security engineers' review of A_{pilot} reduces such errors for the retrained model. We show the misclassification rates in Table IV.

Software engineers do manually mislabel SBRs as NSBRs.

iii. *Probability Ranking*

RQ3: How effective is our model at classifying SBRs that are manually-mislabeled as NSBRs? How much negative impact would training the model on SBRS manually-mislabeled as NSBRs cause on applying our approach?

The x-axis in Figure II shows the probability ranking when the trained model is tested on A_{pilot}. Approximately 25% of the BRs are found to have a greater-than-74.1% probability of being an SBR, and 75% of the BRs are found to be below a 50.0% probability. The model predicts that the BRs fall into two separate groups: one group with high estimated probabilities of being SBRs, and the other group with low estimated probabilities (suggesting BRs in the other group to be NSBRs). The group with the higher estimated probabilities is small relative to the other. This distinction enables security engineers to prioritize their fortification efforts to a small subset of BRs in the BTS.

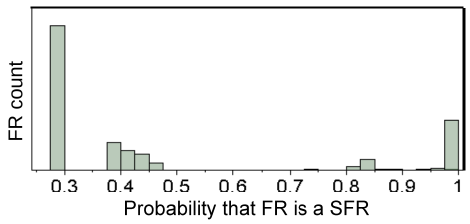

Figure II. Distribution of estimated probabilities from the trained model on A_{pilot}.

The success rate is 68.8% for the trained model and 93.8% for the retrained model as shown in Table IV. The success rates and high precision rates (see Table IV) indicate that the model is effective for classifying unlabeled and manually-mislabeled SBRs. The misclassification rate for the retrained model is lower than the one for the trained model for the training, validation, and test sets, as shown in Table IV. The identification of SBRs mislabeled as NSBRs in A_{pilot} indicates that some of the NSBRs in $A_{feasibility}$ may actually be SBRs. If the trained model misclassified SBRs with a threshold where the probability is greater than 50% in $A_{feasibility,}$ they are considered false positives and the resulting misclassification rate increases. The low misclassification rate from the retrained model indicates that security engineers can more effectively prioritize their fortification efforts to SBRs by using the retrained model.

Table IV. Performance for the trained and retrained models.

			Misclassification rate		
Model (test data set)	Success rate	Precision	Training	Validation	Test
Trained ($A_{feasibility}$)	68.8%	73.2%	27.6%	31.2%	31.2%
Retrained (A_{pilot})	93.8%	60.0%	11.9%	10.4%	6.3%

As described in Section III C ii, we chose probability thresholds based on security engineers' available resources (e.g., afforded inspection efforts). If we raise the threshold to a 97.8% probability for the trained model on the A_{pilot}, then 17.1% of the BRs are found above the threshold. The resulting recall for the SBRs is 77.8% as shown in Table V. That is, security engineers would identify 77.8% of the SBRs in the top 17.1% of the probability ranking. The percentage of NSBRs for the top 17.1% is 63.4%, resulting in a precision of 21.1%. While the FP rate seems high, the resulting count of FPs is fairly low since the threshold restricts the analysis space to only 17.1% of all BRs in the sample. The recall for the retrained model, 75.0% (see Table V), is approximately equal to the recall for the trained model, but the FP rate is only 25%. Additionally, 19.5% of the BRs in the top 17.1% do not have enough information for the Cisco security engineers to determine whether a BR is an SBR.

We tried a threshold of 80.5% in A_{pilot}, and 24.6% of the BRs were located above this threshold. The recall for SBRs here is 88.9% in the top 24.6% of the probability ranking, as shown in Table V. The SBRs below the threshold do not contain diction to indicate that the BRs are likely to be SBRs. The security engineers labeled these BRs as SBRs because their experience with the software indicates that these bugs can be exploited. SBR verbiage is not always suggestive of susceptibility to attack. Additionally, the FP rate for the trained model tested on BCD is 96.2% (see Table V), indicating that security engineers would encounter many NSBRs at the top of the probability ranking.

Table V. Recall for SBRs in the probability ranking.

Model	Test data set	Threshold	Recall	FP
Trained	$A_{feasibility}$	50.0%	64.2%	26.7%
Trained	A_{pilot}	97.8%	77.8%	63.4%
Trained	A_{pilot}	80.5%	88.9%	62.7%
Retrained	A_{pilot}	50.0%	75.0%	25.0%
Trained	BCD	50.0%	30.0%	96.2%

Our natural-language model successfully identifies a high percentage (77%) of SBRs manually-mislabeled as NSBRs by bug reporters. In addition, training our model on SBRs that were manually mislabeled as NSBRs substantially reduces the effectiveness of the model.

iv. Results from Three Additional Systems

RQ4: How effective is our model at classifying unlabeled SBRs in a given system if the model is trained on a BR data set from a different system?

The lift curve (Figure Ic) for the trained model that was tested on the BCD data set does not demonstrate a decrease in SBR identification as the estimated probabilities decrease. The cumulative lift value for the first decile is only 0.09 (see Table III). Furthermore, the precision measured for the trained model is only 3.7%. These results are consistent with those of Anvik et al. [2] where the precision of their algorithm decreased from 64% to 6% when applied to a project whose labeled data were not used to train their model.

The Cisco security engineers analyzed the BRs in A_{pilot} for Systems A, B, C, and D, and identified the types of security bugs. The counts and types are not disclosed to protect company confidentiality. A comparison between Systems A and D showed that the most prevalent security bug type in A was not present in D. Furthermore, the security bug that dominated in D was among the smallest contributors in A. Therefore, training our model on one system's BR data set is likely to be inadequate to classify BRs in another system with different types of security bugs.

The security bug type that dominates in System A comprises approximately half of the security bug types in Systems B and C. The second most predominant security bug type in Systems B and C is the primary security bug type in System D. This analysis shows that the distribution of security bug types between Systems A and Systems B, C, and D are not always similar. The comparison of the security bug types indicates that the verbiage in the SBRs for System A is too dissimilar from the verbiage in Systems B, C, and D to accurately classify SBRs that correspond to different security bug types.

VI. THREATS TO VALIDITY

Our study is representative of only four large software systems and may not necessarily yield the same results for all software systems. The count of BRs in A_{pilot} is smaller than $A_{feasibility}$, but exceeds the minimum count (100) of documents required for statistical modeling, according to SAS [14]. Additionally, BRs are randomly selected from the BTS in an effort to have a similar SBR representation between $A_{feasibility}$ and A_{pilot}. Furthermore, the model's estimated probabilities rely on adequate textual descriptions in the BRs. Bettenburg et al. [3] use a support vector machine (SVM) to determine whether developers agree on BR quality, and they found that their model can correctly predict the developers' BR quality rating. Their results indicate that SVMs can be used to indicate BRs that may require additional details required for a developer to identify and mitigate the problems.

The rating of severity by software engineers could be wrong, which would change the outcome of our analysis. Additionally, the generation of the configuration files is not an objective and repeatable process and so improving or recreating our technique requires human intervention.

VII. RELATED WORK

Various research efforts [3, 6, 8] have been focused on applying text mining on detecting duplicate BRs in BTSs. These efforts can alleviate the work required in triaging BRs to developers. In our work, we also use text mining, but to identify SBRs and prioritize them over NSBRs, addressing a different set of mining requirements.

Cubranic and Murphy [6] use a Bayesian learning algorithm to predict which developer should fix a bug. Their automated technique can reduce the time required by manual analyses to triage BRs. They evaluated their algorithm on the Eclipse BTS and found that the algorithm correctly predicted the most appropriate developer to assess a bug for approximately 30% of the BRs.

Anvik et al. [2] expand the work of Cubranic and Murphy [6] to determine the most appropriate developer for a BR. They use support vector machines to mine the one-line summary and full text description of a BR to create vectors. The vectors are used to predict the software engineer who should fix the bug. Their model reached a precision level of 57% for the Eclipse project and 64% for Firefox. Bettenburg et al. [4] highlight the value of using duplicate bug reports in the Eclipse BTS for training machine-learning models. They observe an accuracy of 65% when predicting which developer should fix a bug. Anvik et al. [2] and Cubranic and Murphy [6] do not train their model to classify SBRs and NSBRs and thus their models may not be applicable for classifying SBRs misclassified as NSBRs.

Jeong et al. [8] use Markov chains to determine which developer should fix a bug. They found that BRs are assigned to developers who then reassign the BRs to other developers. Their graph-based model shows how the reassignment of BRs reveals developer networks. They evaluated their model on the Eclipse and Mozilla projects and found that their model reduces 72% of the reassignments within the developer networks.

Recent research [9] has shown that natural-language information can be used to classify root causes of reported SBRs for Mozilla and Apache HTTP Server . Li et al. [9] collected SBRs from Mozilla and Apache and used a natural-language model to identify the root causes of the security bugs. Based on their results, they determined the semantic security bugs (e.g., missing features, missing cases) comprised 71.9-83.9% of the security bugs. These data provide guidance on what types of tools and techniques that security engineers should use to address most of their security bugs. Their analyses focus on only SBRs that are reported by software and security engineers. In contrast, we apply our model on manually labeled NSBRs to classify SBRs. Additionally, Podgurski et al. [10] use a clustering approach for classifying BRs to prioritize and identify the root causes of bugs, but they do not focus on security bugs.

Runeson et al. [12] use natural-language information to classify duplicate BRs on Sony Ericsson Mobile Communications software. Their model identified approximately 67% of the detectable duplicate BRs. Wang et al. [16] used a natural-language model in addition to execution information of failing tests for BRs to determine which reports are duplicates of pre-existing bug reports in Firefox. They [16] found that when adding execution information as an additional factor to the bug description, they can increase duplicate BR detection from 43-72% to 67-93%. Their results indicate that relying on the text alone of BRs may not be adequate for their predictive models.

VIII. CONCLUSION

BTSs may contain SBRs that are mislabeled by bug reporters as NSBRs. If the security bugs associated with the SBRs escape into the field, then the software can be exploited by attackers. Security engineers may inspect each single NSBR in a BTS to identify SBRs; however, manually inspecting (often thousands of) NSBRs in a BTS is time-consuming and often infeasible, or not even conducted in practice. To address this issue, we propose a novel approach that mines the natural-language text of BRs and constructs a statistical model for predicting which BRs are SBRs. Our approach identified a high percentage (78%) of SBRs mislabeled as NSBRs by bug reporters for a large Cisco software system. To increase the accuracy of our model, software engineers should retrain the model when there are new SBRs being verified by security engineers. But the trained model is not recommended to be applied to software systems in which the SBRs describe different types of security bugs than those that were used to train the model. In summary, our approach effectively automates the identification of SBRs that would otherwise require substantial efforts by security engineers to manually assess each BR in a BTS to determine which BRs are SBRs.

IX. REFERENCES

[1] C. Anley, "Advanced SQL Injection In SQL Server Applications," Next Generation Security Software Ltd, 2002.

[2] J. Anvik, L. Hiew, and G. Murphy, "Who Should Fix This Bug?" *Proc of the ICSE,* pp. 371-380, 2006.

[3] N. Bettenburg, S. Just, A. Schroter, C. Weiss, R. Premraj, and T. Zimmermann, "What Makes a Good Bug Report?" *Proc of the FSE,* pp. 308-318, 2008.

[4] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate Bug Reports Considered Harmful?" *Proc of the ICSM,* pp. 337-345, 2008.

[5] P. Cerrito, *Introduction to Data Mining,* Cary, SAS Institute, Inc., 2006.

[6] D. Cubranic and G. Murphy, "Automatic Bug Triage Using Text Classification" *Proc of the SEKE,* pp. 92-97, 2004.

[7] M. Howard, D. LeBlanc, and J. Viega, *19 Deadly Sins of Software Security: Programming Flaws and How to Fix Them,* Emeryville, McGraw-Hill/Osborne, 2005.

[8] G. Jeong, S. Kim, and T. Zimmermann, "Improving Bug Triage with Bug Tossing Graphs" *Proc of the ESEC-FSE,* 2009.

[9] Z. Li, L. Tan, X. Wang, S. Lu, Y. Zhou, and C. Zhai, "Have Things Changed Now?: An Empirical Study of Bug Characteristics in Modern Open Source Software." *Proc of the 1st Workshop on Architectural and System Support For Improving Software Dependability,* pp. 25-33, 2006.

[10] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang, "Automated Support for Classifying Software Failure Reports" *Proc of the ICSE,* pp. 465-475, 2003.

[11] V. Raghavan and M. Wong, "A Critical Analysis of Vector Space Model for Information Retrieval," *Journal of the American Society for Information Science,* vol. 37, no. 5, pp. 279-287, 1986.

[12] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of Duplicate Defect Reports Using Natural Language Processing" *Proc of the ICSE,* pp. 499-510, 2007.

[13] K. Sarma, *Predictive Modeling with SAS Enterprise Miner,* Cary, SAS Institute, Inc., 2007.

[14] SAS Institute Inc., "Getting Started with SAS 9.1 Text Miner," Cary, NC, 2004.

[15] J. Viega and G. McGraw, *Building Secure Software How to Avoid Security Problems the Right Way,* Boston, Addison-Wesley, 2002.

[16] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An Approach to Detecting Duplicate Bug Reports Using Natural Language and Execution Information" *Proc of the ICSE,* pp. 461-470, 2008.

[17] I. Witten and E. Frank, *Data Mining,* Second ed. San Francisco, Elsevier, 2005.

Tao Xie's work is supported in part by ARO grant W911NF-08-1-0443, and ARO grant W911NF-08-1-0105 managed by NCSU SOSI.