Identifying Security Fault Reports via Text Mining

[1]Michael Gegick, [2]Pete Rotella, [1]Tao Xie
[1]North Carolina State University Department of Computer Science, [2]Cisco Systems
[1]890 Oval Drive, Raleigh, NC, 27695
[2]7200-10 Kit Creek Rd, RTP, NC, 27709
[1]1-919-515-3772, [2]1-919-392-3854

mcgegick@ncsu.edu, protella@cisco.com, xie@csc.ncsu.edu

ABSTRACT

A fault-tracking (bug-tracking) system such as Bugzilla contains fault reports (FRs) collected from various sources such as development teams, test teams, and end-users. Software or security engineers manually analyze the FRs to label the subset of FRs that are security fault reports (SFRs), which indicate a security problem. These SFRs generally deserve higher priority in fault fixing than the not-security fault reports (NSFRs). However, this manual process is time consuming and error-prone (e.g. mislabeling an SFR as an NSFR). To address these important issues, we developed a new approach that applies text mining natural-language descriptions of FRs to train a statistical model on already manually-labeled FRs to identify unlabeled SFRs or SFRs that are manually-mislabeled as NSFRs. A security team can use the model to automate the classification of FRs for large fault databases to reduce the time that they spend on searching for SFRs. We evaluated the model's predictions on a large Cisco software system with over ten million source lines of code. Among a sample of FRs that Cisco software engineers manually labeled as NSFRs, our model successfully classified a high percentage (78%) of the SFRs as verified by a Cisco security team, and predicted their classification as SFRs with a probability of at least 0.98. Our results also indicate that a high percentage (77%) of the SFRs identified by our model is associated with software components that a code-level statistical model predicted to be attack-prone. Such findings provided valuable insights for calling for a future combined approach that exploits both textual information of FRs and code-level information of their associated software components.

1. INTRODUCTION

Software organizations use fault-tracking systems (FTSs) such as Bugzilla[1] to manage fault reports (FRs) collected from various sources including development teams, test teams, and end-users. In an FTS, some FRs are security fault reports (SFRs), whose associated faults are found to be security problems. SFRs generally deserve higher fix priority than not-security fault reports (NSFRs), the subset of FRs that are not believed to have a security impact. Identifying SFRs in an FTS is an important task in security practice. However, manually identifying SFRs (often conducted by software engineers) is not only time-consuming but also error-prone (e.g., mislabeling an SFR as NSFR). Mislabeling SFRs as NSFRs (even to a small extent) can seriously threaten the security assurance of the software, and failure to identify and fix security faults can cause serious damage to software-system stakeholders. For example, an attack on vulnerable credit card data on T.J. Maxx software is estimated[1] to cost the company $4.5 billion.

Software engineers with a general reliability background may realize that some of the FRs in the FTS associated with their software are SFRs, and correctly label them as SFRs. However, they may not recognize the security impact for all the SFRs in the FTS and thus mislabel them as NSFRs. As a consequence, security faults can escape into the field in at least three ways. First, if software engineers perceive a subtle security fault described in an FR as an innocuous not-security fault, then they may assign a low priority to the FR or may not submit the FR to a security team for a review. Second, some security faults described in FRs are associated with recommended mitigations that may be unknown to software engineers. For example, if a SQL parser throws an exception due to input containing a single quote, then a software engineer may filter the input for single quotes. However, attackers can write crafty exploits to circumvent such filters [2]. A security engineer would realize that single quotes can be used in SQL injection attacks and advise that the software engineer limit privileges on a database server and use prepared statements that bind variables as advised by Howard et al. [12]. Third, general reliability problems can also be security problems [22]. For example, a fault that causes a system to crash can also be a denial-of-service security fault if exploited by an attacker.

Indeed, a company can designate security engineers from a security team to manually identify SFRs from an FTS (instead of designating software engineers to do so) or designate security engineers to double check the NSFRs manually-labeled by software engineers. Doing so can alleviate some of the preceding factors to some extent but still cannot eradicate these factors, which are mostly due to human nature. In addition, the large number of FRs in an FTS precludes security engineers from double checking all NSFRs to determine if any SFRs are mislabeled as NSFRs. In summary, there remains a strong need of effective tool support for reducing both human efforts and human mistakes in this process of identifying SFRs in an FTS. With such effective tool support, the security team can elevate the priority of each identified SFR and ensure that the described security fault receives appropriate fortification efforts, and get fixed timely, thus improving the security assurance of the software.

[1] http://www.informationweek.com/news/security/showArticle.jhtml?articleID=199203277

[2] http://www.bugzilla.org/

In this paper, we propose a new tool-supported approach that applies text mining on natural-language descriptions of FRs to learn a statistical model to classify[3] an FR as either an SFR or an NSFR. Although FR submitters may not recognize that the fault they are describing is a security fault, the natural-language description of the fault in the FR may be adequate to indicate that the fault is security-related and thus the FR is an SFR. For example, a FR (that is a SFR) may have the following natural-language description:

"*An attacker can exploit a buffer overflow vulnerability by sending excessive input to the username input field.*"

But if FRs submitter do not realize that the fault is an exploitable buffer overflow, then they may instead write the following text in the FR:

"*The system crashes when receiving excessive input in the username input field.*"

While this second description does not have security-related verbiage, the FR submitters' use of the keywords "crash" "excessive", and "input" can provide valuable and sufficient language context to indicate that the FR is an SFR, so that the security team can timely review the FR and ensure that the described security fault is fortified.

To identify SFRs by exploiting valuable natural-language information in FRs, we propose an approach that learns a natural-language statistical model for classifying an FR as either an SFR or NSFR. We implement our approach based on an industrial text mining tool called SAS Text Miner[4]. 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 SFRs and then applied the model on FRs that were labeled as NSFRs by Cisco software engineers. We also applied the model on FRs 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.

Combining two different predictive models using different sources of information can improve the predictive performance over a single model [24]. The predictive model proposed in our previous work [9] predicts whether a software component is attack-prone based on the following information that is available early in the software life cycle: software metrics including code churn (added and changed SLOC) and warnings produced by a static analysis tool. We applied our natural-language model on FRs associated with attack-prone components to determine whether SFRs are more likely to be associated with components predicted to be attack-prone. Such empirical findings can shed light on whether it is worthwhile of pursuing a future approach that exploits both textual information of FRs and code-level information of their associated software components.

In summary, this paper makes the following main contributions:

- The first approach that learns a natural-language model to automate the prediction of SFRs. We also show how our model can be trained and refined to increase the effectiveness of identifying SFRs mislabeled as NSFRs.
- 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 identify seven times more SFRs from an FTS of one system than randomly selecting FRs from the FTS. Among a sample of FRs that Cisco software engineers originally labeled as NSFRs, our model successfully identified a high percentage (78%) of the SFRs as verified by a Cisco security team, and predicted their classification as SFRs with a probability of at least 0.98.
- The first exploratory empirical study of correlating the results of a code-level predictive model for predicting attack-prone components and our model for predicting SFRs. Our results also indicate that a high percentage (77%) of the SFRs identified by our model is associated with software components that a code-level statistical model predicted to be attack-prone. Such findings provided valuable insights for calling for a future combined approach that exploits both textual information of FRs and code-level information of their associated software components.

The rest of this paper is organized as follows. Section 2 provides background. Sections 3 and 4 detail our approach and case study setup. Section 5 presents the results. Section 6 provides a discussion of the results. Section 7 provides related work. Section 8 discusses the threats to validity, and Section 9 concludes.

2. TEXT-MINING OVERVIEW

Information retrieval (IR) is the discipline of retrieving information, via a query, in unstructured data, especially in textual documents such as an FR as we investigate in this study [11]. Text mining is a type of natural-language processing that parses terms (i.e., words and phrases) from a document to create a term-by-document frequency matrix. Table 1 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 can be used for predictive modeling. The models that use such matrices to represent documents are called vector space models, and are commonly used for IR [17].

Table 1. 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

The investigator performing a textual analysis 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 associated with the documents targeted in a search. If the terms in a document match those in the start list, then those terms are entered into the matrix. A stop list contains terms such as articles, prepositions, and conjunctions that are not used in the analysis. If terms in the stop list match those in a document, then those terms are not entered into the matrix. The synonym list contains terms with the same meanings (e.g., "buffer overflow" and "buffer

[3] We reserve *labeling* for manually identifying FRs as SFRs or NSFRs and *classifying* for model-based identification of SFRs and NSFRs.

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

overrun" have the same meaning). Terms in a synonym list are treated equivalently in a textual analysis. Therefore, a less-used term that is associated with SFRs may be given more weight in the predictive model if the term is synonymous with a term that is often used with SFRs.

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 FR. The formula for calculating the frequency weight of a term with the log function is given in Formula 1, where *i* represents the term, *j* represents the document, and *a* represents the frequency of the entry in the term-by-document matrix [21].

$$L_{ij} = \log_2(a_{ij} + 1) \qquad (1)$$

We use the entropy term weight function to apply higher weights to terms that occur infrequently in the FRs [21]. The formula for calculating the term weight of a term with the entropy function is given in Formula 2, where n represents the number of documents under analysis and p_{ij} represents the frequency of term *i* in document *j* divided by the number of times that term *i* appears in the entire document collection [21].

$$G_i = 1 + \sum \frac{p_{ij} \log_2(p_{ij})}{\log_2(n)} \qquad (2)$$

A statistical model can then estimate the probability that a document belongs in a given category based on the weighted values in the vector. In SAS, four statistical approaches are available for classifying documents: decision trees, regression, neural network, and memory-based reasoning. Classification is a form of prediction [21]. In the context of natural-language processing, the classification involves classifying documents into predefined categories with a pre-classified training set [21].

3. APPROACH

Our approach consists of three main steps. The first step is to obtain a labeled FR data set that contains textual descriptions of faults and labels to indicate whether an FR is an SFR or an NSFR. The labeled FR 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 textual analyses: 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 an FR is an SFR.

3.1 Textual-Data Preparation

The textual-data step (Step 1 in Figure 1) prepares labeled data for building and evaluating our natural-language predictive model. This step includes three sub-steps. First, from an FTS, we obtain FRs that were submitted by stakeholders including development team, test teams, and end-users. Second, we distinguish between SFRs and NSFRs among the obtained FRs. In some commercial software organizations, an FR contains a label field that indicates whether the FR is an SFR. A query on this field in the FTS causes all known SFRs to be returned. If the field is not present in the FTS or an insufficient number of FRs are labeled, then manual effort from software or security engineers is needed to label a subset of all of FRs as SFRs or NSFRs. In the end, we use labeled FRs 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 SFRs and NSFRs. 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 [21], the minimum count of documents required for natural-language modeling is 100.

3.2 Configuration-File Preparation

After we obtain the terms from the FRs, we select terms from them to prepare the start, stop, and synonym lists (Step 2 in Figure 1). To the start list, we manually add terms such as "vulnerability" and "attack" from SFRs. We also include terms (from SFRs) 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 fault. We also include terms from the NSFRs in the stop list. Terms such as "upgrade" and "requirement" may be less likely to be associated with security faults. Stop lists are a standard form for information retrieval [4], and have been successfully used with natural-language modeling [3]. However, both stop lists and start lists are acceptable for information retrieval [21]. SAS Text Miner allows either a start list or stop list to be used in the analysis, but not both. In our approach, we tried each type, and experienced similar prediction results.

To the synonym list, we add synonyms based on examinations of the enumerated terms from SFRs and NSFRs. FR submitters may use security-related verbiage such as "buffer overflow" or "buffer overrun" to describe the same fault. 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 faults.

We next use SAS Text Miner to generate a term-by-document frequency matrix from the terms in FRs based on the start, stop, and synonym lists. The matrix is a quantified format of the natural language descriptions in the FRs. If we include a large number of FRs in the analysis, the term-by-document frequency matrix can become large. A large matrix can hinder the predictive modeling in text mining analysis [7]. 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 [21]. 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 [21]. We use 200 for the value of k for our analyses, which are for prediction instead of clustering.

3.3 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 are trying to predict) is the label (SFR or NSFR) of an FR. 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 FR data set (e.g., FRs without labels) to the model for predicting their labels.

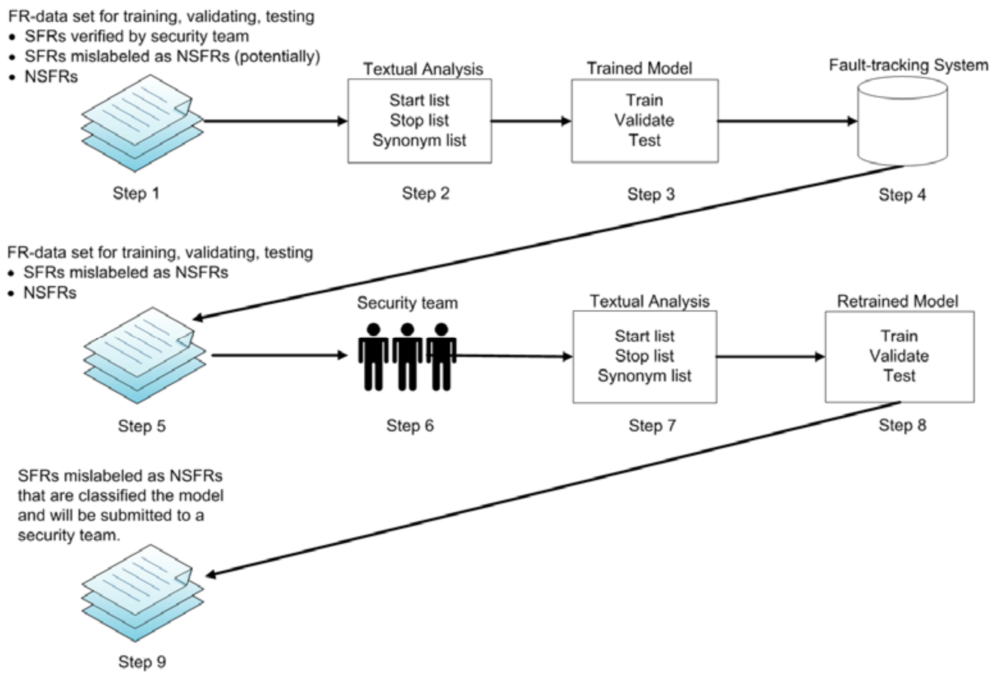
FR-data set for training, validating, testing
• SFRs verified by security team
• SFRs mislabeled as NSFRs (potentially)
• NSFRs
Textual Analysis
Start list
Stop list
Synonym list
Trained Model
Train
Validate
Test
Fault-tracking System
Step 1
Step 2
Step 3
Step 4
FR-data set for training, validating, testing
• SFRs mislabeled as NSFRs
• NSFRs
Security team
Textual Analysis
Start list
Stop list
Synonym list
Retrained Model
Train
Validate
Test
Step 5
Step 6
Step 7
Step 8
SFRs mislabeled as NSFRs that are classified the model and will be submitted to a security team.
Step 9

Figure 1. Summary of approach.

We next describe the training, validation, and test data sets used for training the model, the application of our trained model on new FRs, and retraining of the model with corrected mislabeling (when the initial training data have mislabeled data).

3.3.1 Training, Validation, and Test Data Sets

First, we train, validate, and test the model using the FR data set that we earlier prepared (Step 3 in Figure 1), and divide the FR data set into three smaller data sets: the training, validation, and test data sets [19]. 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 FRs allocated to the training, validation, and test data sets are 60%, 20%, and 20%, respectively, as recommended by SAS [21].

Cubranic and Murphy [8] found that the correct predictions increase from 27% to 30% as the training set is increased from 50% to 90% of the data set size. Their results indicate that the percentage of the training data set does not greatly influence their predictions. Their analyses did not appear to include a validation set.

3.3.2 Application of Trained Model on new FRs

Given new FRs (e.g., FRs without labels) (Step 4 in Figure 1), our built predictive model then estimates the probability that an FR is an SFR. In our setting, the probability ranking is a list of FRs sorted in descending order of the estimated probability of being an SFR. A security team can start their assessments of the FRs at the top of the probability ranking and continue until they reach a pre-defined probability threshold. The threshold indicates that SFRs with probabilities below the threshold may exist, but that there are only few of them.

We determine the probability threshold with the following analysis approach. We first assess the probabilities of SFRs. If all the SFRs have higher probabilities than the NSFRs, we assign the threshold as the lowest probability associated with an SFR. If some NSFRs have higher probabilities than some SFRs, the threshold must be made based on the security team's available resources. Based on the results from the test set, we can determine the lowest estimated probabilities assigned to SFRs. The security team should look at the lowest probability of an SFR in the test set and then match that probability to the probability ranking of the FRs that they intend to assess. If there are too many FRs above that probability to assess, then the security team should use the threshold with the next to lowest probability.

3.3.3 Model Retraining with Corrected Mislabeling

One inherent challenge in our research context is the (un)certainty of the labeling of SFRs (by software engineers) initially used for training the model. The first author's empirical investigations (with security teams in software organizations other than Cisco) have revealed that SFRs are sometimes mislabeled as NSFRs by software engineers. If we train the model on SFRs mislabeled as NSFRs, the model may classify security-related verbiage as not security-related, and, as a consequence, incorrectly predict that an SFR with security-related verbiage is an NSFR. Therefore, the model's accuracy would likely be improved if security engineers from a security team review each FR used to train the model to ensure that its label is correct. To address this issue, we select a subset of the NSFRs (Step 5 in Figure 1) from the FTS. Then, we submit the NSFRs to a security team (Step 6 in Figure 1) for them to check for mislabeling. If any true SFRs exist among the mislabeled NSFRs (which are thus mislabeled) then we add or subtract terms (Step 7 in Figure 1) from the original configuration-files that appear in the SFRs and NSFRs that were reviewed by the security team. We then retrain the model with the subset of NSFRs that is now correctly labeled (Step 8 in

Figure 1). The verbiage between SFRs that contain explicit security verbiage (e.g., attack) may be different than SFRs that are mislabeled as NSFRs which do not contain explicit security verbiage. By training the model on SFRs that are mislabeled as NSFRs (in addition to true NSFRs) the model can identify SFRs with terms that software engineers are likely to use to describe security faults when they do not realize the problem is security-related (Step 9 in Figure 1).

3.3.4 Summary of Approach

In summary, there exist two types of SFRs in an FTS. The first type of SFR includes FRs that contain security-related diction (e.g., "attack"). These SFRs are either recognized by software engineers or are FRs that software engineers submit to a security team to verify that the FR is a SFR. These SFRs may be manually labeled or unlabeled by software engineers in an FTS. The second type of SFR is an FR that describes a security fault, but is mislabeled as an NSFR because software engineers do not recognize the fault as a security fault. The second type of SFR does not include the security-related diction that exists in the first type of SFR, but likely includes other verbiage such as "crash," or "excessive."

To build a predictive model that identifies SFRs mislabeled as NSFRs, we first need to build the Trained Model (see Figure 1). The Trained Model classifies SFRs that are recognized by software and security engineers. The terms from these SFRs are incorporated into the configuration files to determine which terms indicate security faults. These configuration files are important for making the Trained Model be a security-related model. If the SFRs in the FTS are unlabeled, then a security team can use the Trained Model to identify the unlabeled SFRs.

When we apply the Trained Model to the FTS, we will likely obtain both types of SFRs. Obtaining the second type of SFR and having the security team verify the FR as an SFR enables us to create a new training set containing only SFRs mislabeled as NSFRs and true NSFRs. We modify the configuration files from the Trained Model to more closely resemble the terms used in SFRs mislabeled as NSFRs. The new training set is used to train the Retrained Model (see Figure 1) to specifically classify SFRs that are mislabeled as NSFRs and is likely to be more apt at classifying the second type of SFR than the first type of SFR. A security team can use the Retrained Model to automate the classification of SFRs mislabeled as NSFRs (the second type of SFR) in the FTS.

4. 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.

4.1 Four Cisco Software Systems

We analyzed four large Cisco software systems, referred to as Systems A, B, C, and D. Details of these systems are confidential. 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 FTS contains all FRs associated with these software systems, and these reports document both faults and failures in the software systems. Each FR contains a field that is manually filled (initially by software engineers) to label the FR as an SFR. A security team can then evaluate a labeled SFR to either verify that the FR is in fact an SFR, or, if not, reset the field to indicate an NSFR. Each FR also contains a summary text field and a larger description text field. Our natural-language analyses focus on these two text fields of FRs 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.

4.2 Research Questions

In our studies, we address the following research questions:

- RQ1: How effective is our model at classifying unlabeled SFRs of a given system if the model is trained on an FR data set from the *same* system?
- RQ2: Do software engineers fail to recognize that some FRs are SFRs?
- RQ3: How effective is our model at classifying SFRs that are manually-mislabeled as NSFRs? And, how much negative impact would training the model on SFRS manually-mislabeled as NSFRs cause on applying our approach?
- RQ4: How effective is our model at classifying unlabeled SFRs in a given system if the model is trained on an FR data set from a *different* system?
- RQ5: How well do the classification results produced by our FR-level model align with the attack-prone-component-classification results produced by our previous model based on code-level metrics [9]? Can the two models be used sequentially to improve the SFR identification rate?

The answer to RQ1 helps us to assess the effectiveness of our approach when applied to cases where the submitter describes a security problem in the FR description, but does not label the FR.

The answer to RQ2 helps us to determine whether the Cisco security team should review the Cisco FTS for SFRs that are manually-mislabeled as NSFRs by software engineers.

The answer to RQ3 helps us to determine whether our approach is effective in automatically classifying SFRs that software engineers manually-mislabel as NSFRs. If such FRs exist, then we should include the terms associated with these SFRs 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 identifying SFRs in other systems that the model was not trained on. The results can indicate whether software engineers can obtain assistance from our approach in automatically labeling all the unlabelled FRs 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.

The answer to RQ5 helps us to assess the effectiveness for prioritizing security inspections of FRs based on sequentially applying the results of the attack-prone component predictive model and the natural language predictive model.

The metrics used to measure the effectiveness of the model are the quantity of SFRs identified by the model and the quantity of SFRs identified by randomly selecting FRs from a FTS. We use a lift curve to represent the benefit of using the model.

4.3 RQ1 Study Setup

We first queried the Cisco FTS for manually-labeled SFRs associated with System A for the past four years. Next, we randomly sampled System A's FRs that were manually-labeled as NSFRs by software engineers. The samples of SFRs and NSFRs are equal in counts to provide the model with enough data to classify both SFRs and NSFRs accurately. We call the data set of SFRs and NSFRs for System A the **$A_{feasibility}$** 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 intent of the trained model is to predict SFRs that were previously submitted by software engineers. The results would not indicate whether the model correctly predicts an SFR that was mislabeled as an NSFR. The initial results are used to calibrate the model and provide an assessment of the predictive power of natural-language FR descriptions.

4.4 RQ2 Study Setup

We randomly sampled FRs (from System A) that were manually-labeled as NSFRs by software engineers. We call this data set **A_{pilot}**. We applied the trained model on A_{pilot} to estimate the probability that a manually-labeled NSFR is an SFR. We then submitted A_{pilot} to the security team for them to review the same content (that the model used for prediction) to determine whether any of the manually-labeled NSFRs are actually SFRs. We did not reveal the estimated probabilities to the security team to reduce potential bias in their analyses. Based on prior discussions with the security team, we estimated that security engineers would require approximately 175 person-hours to analyze A_{pilot} and determine whether the manually-labeled NSFRs are actually SFRs. At least two members from the security team independently reviewed each FR. If two security engineers disagreed on their evaluations of a manually-labeled FR, 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.

A study where neither the researchers nor participants know which data comprises the experimental group and which comprises the control group is a double-blind case study [15]. Our analysis of NSFRs with the Cisco security team is a double-blind case study. The experimental group contained manually-labeled NSFRs that have high probabilities of being SFRs, according to the model. The control group contains the manually-labeled NSFRs that have low probabilities of being a SFR. We did not provide the security team with probabilities estimated by the model to reduce bias in their analyses. This is the first blind. The second blind is that we did not pre-select FRs before running the model. Reading the FRs before running the model may have resulted in selecting FRs that closely resemble SFRs the model might correctly classify.

Having the security team evaluate each FR in A_{pilot} enables us to be certain of the label of each FR in A_{pilot}. We then retrain the model on A_{pilot} to determine whether a model trained on SFRs mislabeled as NSFRs can be useful for the specific purpose of identifying those SFRs that are manually-mislabeled as NSFRs. 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 team. We call the model that is trained on A_{pilot} the "**retrained**" model.

4.5 RQ3 Study Setup

The security faults associated with System A may be specific for that software system. Additionally, the software engineers for System A may have different writing styles and diction for describing faults than software engineers from other software systems. To investigate these possibilities, we randomly sampled six months of fault reports from Systems B, C, and D and combined them into one data set that we call **BCD**. We tested if the trained model, constructed using data from System A, can effectively identify SFRs 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 2 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 2. 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}

4.6 RQ4 Study Setup

In our previous work [9], we created a statistical model that predicts attack-prone components. We refer to this model as the *component model* in this paper. In our previous study, this component model identified 76% of System A's attack-prone components. At Cisco, a component is a group of functionally-related source code files. The independent variables of this model are the count of warnings produced by static analysis tools, the count of code churn (i.e., added and changed SLOC) and size, and the count of faults found by manual inspections. The dependent variable is a binary value indicating whether the software component is attack-prone or not. An attack-prone component in our previous study is a component that had at least one known SFR. The results of the component model are shown in Figure 2. The areas of the regions in Figure 2 are proportional to the count of components in those regions. The true positive (TP) region represents components that the model correctly predicted as attack-prone. The false positives (FP) region represents components that the model predicted to be attack-prone, but there were no known SFRs associated with the components. The false negative (FN) region represents components that are not predicted to be attack-prone by the model, but do have at least one SFR (as verified by the security team) associated with them. The true negative (TN) region represents components that are correctly predicted to be not attack-prone, since they have no known SFRs associated with them.

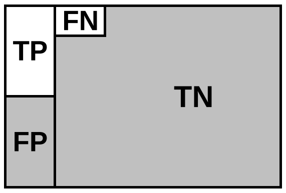

Figure 2. A visualization of System A components as classified by the attack-prone component predictive model where non-shaded regions indicate attack-prone components.

In this study for RQ4, we examine how the SFRs in A_{pilot} are distributed over the regions (see Figure 2) produced by the component model. Each FR pertains to a specific component, so there is a direct link between the predictions of the two models. We perform an empirical evaluation to determine if most of the SFRs identified by the natural-language model are associated with components predicted to be attack-prone. If so, then the FRs associated with predicted attack-prone components can be focused areas for software engineers to invest their FR-labeling effort (for preparing the FR data set required for training the natural-language model) or for security engineers to invest their FR-reviewing effort (for finding manually-labeled NSFRs that are actually SFRs). An equal number of FRs were randomly selected from the TP, FP, and TN regions of the component model to create A_{pilot}. The components in the FN region are excluded from the analysis because there are too few components in the FN region for statistical reliability.

5. RESULTS

In the feasibility study for the $A_{feasibility}$ data set, we found that the neural-network model identified at least five percent more SFRs than the decision tree, regression, or memory-based reasoning predictive models, and we therefore use a neural network for our predictions. We also found that a model with a start list and synonym list identified approximately the same count of manually-labeled SFRs as a model with a stop list. We chose to use a start list and synonym list for our analyses because we suspect that continually updating the start list is more feasible for a limited number of security faults than managing a large stop list.

If the model classifies an SFR as an NSFR, or if the model classifies an NSFR as an SFR, then the result is a misclassification. As discussed in Section 4.6, we now define the correct classifications and misclassifications for the natural-language model. A true positive (TP) is a verified (by a security engineer) SFR that is correctly predicted by the model. A false positive (FP) is a verified NSFR that is incorrectly predicted to be an SFR. A false negative (FN) is a verified SFR that is incorrectly predicted to be an NSFR. A true negative (TN) is a verified NSFR that is correctly predicted to be an NSFR. The success rate of the model is the number of verified classifications divided by the total number of classifications [24]. Model precision is the proportion of correctly classified SFRs divided by the sum of SFRs and NSFRs that have been determined by the model to be SFRs (i.e., exceeding a minimum probability). In our setting, recall is the percentage of verified SFRs that the model predicts (above a minimum probability). The formulas for the success rate, precision, and recall are provided below.

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

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

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

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

5.1 Lift Curves and Tables

We measure the effectiveness of the model with lift curves, which quantify how much the model improves the rates of identifying SFRs, compared to randomly selecting and analyzing FRs from the FTS. The lift curves for the trained model tested on $A_{feasibility,}$ A_{pilot}, and BCD, and the retrained model tested on A_{pilot} are shown in Figure 3. We explain the details of these results in later subsections. The lift curve x-axis represents the FRs sorted in descending order of likelihood of being an SFR, as predicted by the model, and then divided into ten deciles, where the leftmost decile contains the FRs with the highest likelihood of being an SFR. The y-axis represents the percentage of total SFRs (called the “cumulative” percentage) contained in a given decile that is identified by the model. The lift curves are cumulative in the sense that the counts of SFRs and FRs are aggregated within each of the ten deciles. An accurate model is one in which the highest SFR rate occurs in the first decile, the second highest in the second decile, and so on. The horizontal dashed lines in Figure 3 are baselines that represent the SFR rates for the data set’s deciles, assuming a model is not used and that SFRs are found randomly. The overall SFR rate is equal to the count of SFRs divided by the count of all FRs. For a given decile, the difference between the cumulative SFR rate that is derived from the model and the baseline rate represents the effectiveness of the model’s predictions for that decile. The specific values on the y-axis in Figure 3 are not disclosed in order to conceal the Cisco SFR rate.

In Table 3, we show the cumulative SFR lift values for each decile for each data set shown in Figure 3. The cumulative lift value in the first decile is equal to the SFR rate of the model in the first decile divided by the overall SFR rate [19]. The cumulative lift value in the second decile is equal to the cumulative SFR rate of the model divided by the overall SFR rate for the second decile, and so on. The rest of this section compares the model’s predictions to randomly selecting FRs from the FTS.

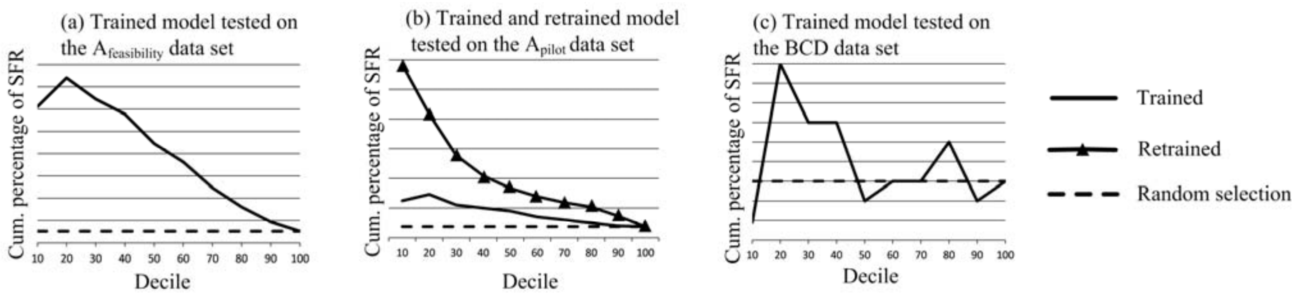

Figure 3. Lift curves for case study results.

5.1.1 Identifying SFRs in System A

RQ1: How effective is our model at classifying unlabeled SFRs of a given system if the model is trained on an FR data set from the same system?

The lift curve for the model that is trained, validated, and tested on $A_{feasibility}$ (see Figure 3a) indicates that the chance of finding an SFR generally decreases as the model's estimated probabilities decrease. Although the first decile (associated with the highest probabilities) contains some SFRs, the highest percentage of SFRs 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 3 for $A_{feasibility}$. The results indicate that a security team would identify 1.53 times more SFRs in the first decile with the model than by randomly selecting FRs from the FTS. Therefore, this result shows that the verbiage in SFR text fields can be successfully used to predict other unlabeled SFRs. While this lift is low, the lift is nevertheless positive, providing enough justification to continue our analyses on SFRs mislabeled as NSFRs.

Table 3. Cumulative lift values for the 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 predicting SFRs that software engineers realize are true SFRs.

5.1.2 Identification of SFRs Mislabeled as NSFRs

RQ2: Do software engineers fail to recognize that some FRs are SFRs?

The security team verified that some[†] of the FRs that were labeled as NSFRs by software engineers are actually SFRs. Figure 3b shows the lift curve when the trained model (i.e., trained on $A_{feasibility}$) is used to identify SFRs in A_{pilot}. The cumulative lift value for the first decile is 3.33 (see Table 3), indicating that security engineers would identify 3.33 times more SFRs that are mislabeled as NSFRs by using the model than they would by randomly selecting FRs from the FTS.

The model identified several[†] types of security faults demonstrating that the model does not identify only one type of security fault. One of the SFRs identified in the analysis was also reported in the field, thereby indicating that the model can be used to identify FRs that are found both internally and externally to Cisco. If the SFRs discovered by the security team had not already been fixed, it would have received either an elevated priority and 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 3b). This result indicates that as the estimated probabilities decrease, the likelihood of an SFR also decreases. The largest cumulative lift value in our case studies, 7.00, is in the first decile for the retrained model. The language used to describe the SFRs in $A_{feasibility}$ may closely resemble the SFRs mislabeled as NSFRs in A_{pilot}, which is likely to be responsible for improving the accuracy of the retrained model. Similarly, the NSFRs in A_{pilot} may have resemblance to the NSFRs in $A_{feasibility}$. For each run of the model, security engineers can add examined FRs to the training and validation sets to improve the model's accuracy. Additionally, a security team can add or subtract terms (collected from SFRs identified by the team) 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 FR in the data set. We are therefore certain which FRs are SFRs and which are NSFRs. This certainty improves the retrained model's results over the trained model's results with two reasons. First, the certainty can improve the model training and validation compared to other training and validation data sets that may have SFRs mislabeled as NSFRs. Training the model with SFRs mislabeled as NSFRs can result in the model's misclassifying SFRs as NSFRs. Second, the certainty improves the accuracy of the model in the test data

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

set in A_{pilot} compared to evaluations in other test data sets. If an SFR is mislabeled as an NSFR by a software engineer in $A_{feasibility}$, but the model predicts the FR to be an SFR, then the result is a false positive. The cumulative lift values are decreased due to instances where the model is correct, but the labeling of the FR is incorrect. The security engineers' review of A_{pilot} reduces such errors for the retrained model. We show the misclassification rates in Section 5.3.

Software engineers do manually mislabel SFRs as NSFRs.

5.1.3 Probability Ranking

RQ3: How effective is our model at classifying SFRs that are manually-mislabeled as NSFRs? And, how much negative impact would training the model on SFRS manually-mislabeled as NSFRs cause on applying our approach?

The x-axis in Figure 4 shows the probability ranking when the trained model is tested on A_{pilot}. Approximately 25% of the FRs are found to have a greater-than-74.1% probability of being an SFR, and 75% of the FRs are found to be below a 50.0% probability. The model predicts that the FRs fall into two separate groups: one group with high estimated probabilities of being SFRs, and the other group with low estimated probabilities (suggesting FRs in the other group to be NSFRs). The group with the higher estimated probabilities is small relative to the other. This distinction enables a security team to prioritize their fortification efforts to a small subset of FRs in the FTS.

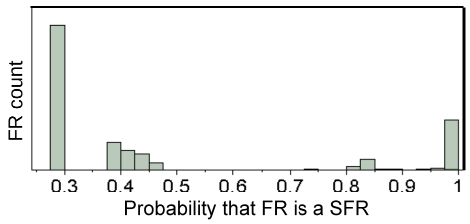

Figure 4. Distribution of estimated probabilities from the trained model on A_{pilot}.

The success rate is 68.8% for the trained model and 93.8% for the retrained model as shown in Table 4. The success rates and high precision rates (see Table 4) indicate that the model is effective for identifying unlabeled and manually-mislabeled SFRs. The misclassification rate for the retrained model is lower than for the trained model for the training, validation, and test sets, as shown in Table 4. The identification of SFRs mislabeled as NSFRs in A_{pilot} indicates that some of the NSFRs in $A_{feasibility}$ may actually be SFRs. If the trained model misclassified SFRs 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 a security team can more effectively prioritize their fortification efforts to SFRs by using the retrained model.

Table 4. 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%

If we raise the threshold to a 97.8% probability for the trained model on the A_{pilot}, then 17.1% of the SFRs are found above the threshold. The resulting recall for the SFRs is 77.8% as shown in Table 5. That is, security engineers would identify 77.8% of the SFRs in the top 17.1% of the probability ranking. The percentage of NSFRs 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 FRs in the sample. The recall for the retrained model, 75.0% (see Table 5), is approximately equal to the recall for the trained model, but the FP rate is only 25%. Additionally, 19.5% of the FRs in the top 17.1% did not have enough information for the Cisco security team to determine if an FR is an SFR.

We tried a threshold of 80.5% in A_{pilot}, and 24.6% of the SFRs were located above this threshold. The recall for SFRs here is 88.9% in the top 24.6% of the probability ranking, as shown in Table 5. The SFRs below the threshold do not contain diction to indicate that the FRs are likely to be SFRs. The security engineers labeled these FRs as SFRs because their experience with the software indicates that these faults can be exploited. SFR 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 5) indicating that a security team would encounter many NSFRs at the top of the probability ranking.

Table 5. Recall for SFRs 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 SFRs manually-mislabeled as NSFRs by software engineers. Also, training our model on SFRs that were manually mislabeled as NSFRs substantially reduces the effectiveness of the model.

5.1.4 Results from Three Additional Systems

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

The lift curve (Figure 3c) for the trained model that was tested on the BCD data set does not demonstrate a decrease in SFR identification as the estimated probabilities decrease. The cumulative lift value for the first decile is only 0.09 (see Table 3). Furthermore, the precision measured for the trained model is only 3.7%. These results are consistent with those of Anvik et al. [3] 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 team analyzed the FRs in A_{pilot} for Systems A, B, C, and D, and identified the types of security faults. The counts and types are not disclosed to protect company confidentiality. A comparison between Systems A and D showed that the most prevalent security fault type in A was not present in D. Furthermore, the security fault that dominated in D was among the smallest contributors in A. Therefore, training our model on one system's FR data set is likely to be inadequate to predict FRs in a system with different types of security faults.

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

The security faults and natural-language descriptions of SFRs among different systems are too different for our natural-language model to be successfully applied to a different system from the one whose FR data are used to train the model.

5.2 Fit Statistics

A good statistical practice is to gauge the uncertainties in the estimates of a predictive model. The average squared error (ASE) is the average of the square of the difference between the predicted outcome and the actual outcome [19]. The ASE is calculated during the model training for the validation data set [19]. The weights of the terms with the smallest error are used in the final predictive equation [19]. We report the ASE in the training, validation, and test sets for the trained model on $A_{feasibility}$ and the retrained model on A_{pilot} in Table 6. The ASE for the retrained model is approximately three times less than the trained model for the training, validation, and test sets. These results indicate there is more statistical certainty in the estimated probabilities for the retrained model than for the trained model. The higher statistical certainty in the retrained model may be accounted for by the certainty that each SFR is known in A_{pilot} from the security team analysis as mentioned in Section 5.1.2.

Table 6. Fit statistics for the trained and retrained model.

Model	ASE training	ASE validation	ASE test	AIC	SBC
Trained	0.19	0.22	0.21	562.57	666.02
Retrained	0.07	0.07	0.05	83.07	115.66

We also measure the fit statistics of the model with the Akaike Information Criterion (AIC) and the Schwarz Bayesian Criterion (SBC). Both fit statistics assess the model parameters, error sum of squares, and the count of FRs. The AIC and SBC penalize the model for additional parameters [20]. The smaller the values of the AIC and SBC, the better the statistical fit of the model [19]. The AIC and SBC values decrease by approximately 85% in the retrained model as shown in Table 6 indicating that the parameters for the model are a better fit in the retrained than the trained model. We expect to have more reliable predictions with retrained model due to the lower ASE.

5.3 Empirical Evaluation of Sequential Modeling

RQ5: How well do the classification results produced by our FR-level model align with the attack-prone-component-classification results produced by our previous model based on code-level metrics [9]? Can the two models be used sequentially to improve the SFR identification rate?

We examined how the SFRs discovered in A_{pilot} by the natural-language model are distributed over the TP, FP, and TN regions of the component model discussed in Section 4.5. The largest percentage of SFRs, 44%, is associated with software components in the FP region (see Figure 2). The components in this region were not associated with any SFRs until the security team analyzed A_{pilot}. The component model's FP predictions that were originally considered erroneous are actually correct, as determined by the security team's reviews of FRs identified by our natural-language model. The agreement between the models' results suggests that a subset of components that software engineers and a security team had previously thought was not attack-prone actually is attack-prone. These newly discovered attack-prone components should be targeted for security fortification, including threat modeling, penetration testing, and other security-specific inspections.

Approximately 33% of the SFRs identified by the natural-language model are associated with the component model's TP region. Therefore, the results show that 77% (44% from the FP region, 33% from the TP region) of the SFRs are associated with components that the component model predicted to be attack-prone. The good agreement between the component and natural-language predictive models indicates that a security team should apply the two predictive models sequentially. First, the security team should apply the component model to predict the attack-prone components. Then, the FRs associated with predicted attack-prone components can be focused areas for software engineers to invest their FR-labeling effort (for preparing the FR data set required for training the natural-language model) or for security engineers to invest their FR-reviewing effort (for finding manually-labeled NSFRs that are actually SFRs).

Approximately 23% of the SFRs discovered are associated with components in the component model's TN region. However, the natural-language model successfully predicted the existence of all the SFRs (as verified by the security team) in the TN region. While there is this disagreement between the two models in the TN region, this region contains the lowest fraction of the SFRs.

If we removed all the FRs from the TN region, then we would have removed 26.3% of false positives above the 80.5% threshold (see Section 5.1.3). The decision to include the component model's TN region in the FR analysis is a risk management decision. If the security team has the resources to review the false positives (as determined by the natural-language model) from the TN region of the component model, then can identify 88.9% of the SFRs. If the security team does not have the resources to review false positives from the TN region, they can review FRs associated with only the TP and FP regions and obtain 77% of the SFRs.

SFRs manually-mislabeled as NSFRs are more likely to be associated with components predicted to be attack-prone. Security engineers should prioritize FR-reviewing effort first for FRs associated with predicted attack-prone components.

6. DISCUSSION

We observe that most of the SFRs identified by the trained model have estimated probabilities in the second decile (see Figures 3a-3c) for the $A_{feasibility}$, A_{pilot}, and BCD data sets. The FRs that were predicted to be SFRs, but are actually NSFRs in the first decile, contain diction that is used in SFRs, but is also commonly used in NSFRs. We analyzed these NSFRs and found three main reasons why they were assigned a high probability of being an SFR by the models. First, the NSFRs are general reliability or performance problems that fail in the same way as the security faults, but are not remotely trigger-able by an attacker. Second, the not-security faults described in these NSFRs were found not to be severe enough to cause a security impact on the software system. Third, some NSFRs explained that a security feature was not working correctly, rather than that a true security fault existed. For example, the permissions were set too high or too low for an object. While some of the FRs in the first decile are SFRs, there exist more NSFRs that describe similar, but not exploitable, faults.

7. RELATED WORK

Cubranic and Murphy [8] investigated IR in the context of FRs of open source FTSs. They use a Bayesian learning algorithm to predict which developer should fix a fault. Their automated technique can reduce the time required by manual analyses to triage the FRs. They evaluated their algorithm on the Eclipse FTS and found that the algorithm correctly predicted the most appropriate developer to assess a fault for approximately 30% of the FRs.

Anvik et al. [3] expand the work of Cubranic and Murphy [8] to determine the most appropriate developer for an FR. They use support vector machines to mine the one-line summary and full text description of an FR to create vectors. The vectors are used to predict the software engineer who should resolve the fault. Their model reached a precision level of 57% for the Eclipse project and 64% for Firefox. Bettenburg et al. [6] further the value of using duplicate bug reports in the Eclipse FTS for training machine-learning models. They observe an accuracy of 65% when predicting which developer should fix a fault. Anvik et al. [3] and Cubranic and Murphy [8] do not train their model to classify SFRs and NSFRs and thus their models may not be applicable for identifying SFRs misclassified as NSFRs.

Jeong et al. [13] use Markov chains to determine which developer should fix a fault. They found that FRs are assigned to developers who then reassign the FR to other developers. Their graph-based model shows how the reassignment of FRs 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 has shown that natural-language information can be used to classify root causes of reported SFRs for Mozilla and Apache HTTP Server [14]. Li et al. [14] collected SFRs from Mozilla and Apache and used a natural-language model to identify the root causes of the security faults. Based on their results, they determined the semantic security faults (e.g., missing features, missing cases) comprised 71.9-83.9% of the security faults. These data provide guidance on what types of tools and techniques a security team should use to address most of their security faults. Their analyses focus on only SFRs that are reported by software and security engineers. In contrast, we apply our model to NSFRs to identify security faults. Additionally, Podgurski et al. [16] use a clustering approach for classifying FRs to prioritize and identify the root causes of faults, but they do not focus on security faults.

Runeson et al. [18] use natural-language processing to identify duplicate FRs on Sony Ericsson Mobile Communications software. Their model identified approximately 67% of the detectable duplicate FRs. Wang et al. [23] used a natural-language model in addition to execution information of failing tests for FRs to determine which reports are duplicates of pre-existing fault reports in Firefox. Wang et al. [23] found that when adding execution information as an additional factor to the fault description, they can increase duplicate FR detection from 43-72% to 67-93%. Their results indicate that relying on the text alone of FRs may not be adequate for their predictive models.

8. 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. Also, the count of FRs in A_{pilot} is smaller than $A_{feasibility}$, but we exceeded the minimum count (100) of documents required for statistical modeling, according to SAS [21]. Additionally, the FRs are randomly selected from the FTS in an effort to have a similar SFR representation between $A_{feasibility}$ and A_{pilot}. Furthermore, the model's estimated probabilities rely on adequate textual descriptions in the FRs. Bettenburg et al. [5] use an SVM to determine whether developers agree on FR quality, and they found that their model can correctly predict the developer's FR quality rating. Their results indicate that SVMs can be used to indicate FRs that may require additional details required for a developer to identify and mitigate the problems. Such a model can be used to mitigate the limitations of low-quality FRs for natural-language models. Details of the fit statistics are available in our accompanying technical report [10].

The selection of terms to include in our start list and synonym list is made subjectively. We do not use an objective and repeatable means to define a start list and synonym list but once made the lists are available for any additional model we create. Finally, security faults have been shown to be rare events in software systems [1]. We have few security faults in our data set. The scarce data reduce the certainty of statistical analyses.

9. CONCLUSION

Fault-tracking systems (FTSs) may contain SFRs that are either not labeled or are mislabeled as NSFRs. If the security faults associated with the SFRs escape into the field, then the software can be exploited by attackers. To address this issue, we propose a novel approach that mines the natural-language text of FRs and constructs a statistical model for predicting which FRs are SFRs. Our approach identified a high percentage (78%) of SFRs mislabeled as NSFRs by software engineers for a large Cisco software system. To increase the accuracy of our model, the security team should retrain the model when there are SFRs being newly verified by the security team. But the trained model is not recommended to be applied to software systems in which the SFRs describe different types of security faults than those that were used to train the model. Lastly, an empirical evaluation of using natural-language and component models sequentially indicates that we can increase the accuracy of both models' predictions. In summary, our approach effectively automates the identification of SFRs that would otherwise require a substantial effort by a security team to manually assess each FR in an FTS to determine which FRs are SFRs.

10. ACKNOWLEDGEMENTS

We are very grateful for the participation from the Cisco SEO and PSIRT security teams, and feedback (on earlier versions of this paper) by Yeva Byzek, Jeff Curka, Robert Lake, Ramya Ravichandar, and Wesley Zhang.

11. REFERENCES

[1] Alhazmi O. H., Y. K. Malaiya, and I. Ray, "Measuring, analyzing and predicting vulnerabilities in software systems," *Computers & Security*, vol. 26, no. 3, pp. 219-228, May 2006.

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

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

[4] Baeza-Yates R. A. and B. A. Ribeiro-Neto, *Modern Information Retrieval*, 1999.

[5] Bettenburg N., S. Just, A. Schroter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?," In Proc. of *FSE*, pp. 308-318, 2008.

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

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

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

[9] Gegick M., P. Rotella, and L. Williams, "Predicting Attack-prone Components," In Proc. of *ICST*, pp. 181-190, 2009.

[10] Gegick M., P. Rotella, and T. Xie, "Identifying Security Fault Reports via Text Mining," North Carolina State University, ftp://ftp.ncsu.edu/pub/unity/lockers/ftp/csc_anon/tech/2009/TR-2009-17.pdf, 2009.

[11] Greengrass E., *Information Retrieval: A Survey*, Baltimore County, University of Maryland, 2000.

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

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

[14] Li Z., 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.," In Proc. of *1st Workshop on Architectural and System Support For Improving Software Dependability*, pp. 25-33, 2006.

[15] Motulsky H., *Intuitive Biostatistics*, New York, Oxford University Press, 1995.

[16] Podgurski A., D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang, "Automated support for classifying software failure reports," In Proc. of *ICSE*, pp. 465-475, 2003.

[17] Raghavan V. 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.

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

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

[20] SAS Institute Inc., "SAS Help and Documentation," SAS Institute, Inc., Cary, 2003.

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

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

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

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