Toward Non-security Failures as a Predictor of Security Faults and Failures

Michael Gegick[1], Pete Rotella[2], and Laurie Williams[1]

[1]Department of Computer Science, North Carolina State University
890 Oval Drive, Raleigh, NC, USA
[2]Cisco Systems, Inc.
7025-6 Kit Creek Rd, Research Triangle Park, NC, USA
{mcgegick, lawilli3}@ncsu.edu, protella@cisco.com

Abstract. In the search for metrics that can predict the presence of vulnerabilities early in the software life cycle, there may be some benefit to choosing metrics from the non-security realm. We analyzed non-security and security failure data reported for the year 2007 of a Cisco software system. We used non-security failure reports as input variables into a classification and regression tree (CART) model to determine the probability that a component will have at least one vulnerability. Using CART, we ranked all of the system components in descending order of their probabilities and found that 57% of the vulnerable components were in the top nine percent of the total component ranking, but with a 48% false positive rate. The results indicate that non-security failures can be used as one of the input variables for security-related prediction models.

Keywords: attack-prone, classification and regression tree.

1 Introduction

In the search for metrics that can predict the presence of vulnerabilities, there may be some benefit to choosing metrics from the non-security realm. Metrics can be used as input variables in statistical models to identify which software components[1] are most likely to be attacked. Such a statistical model can afford for security engineers to prioritize their efforts to the highest risk components.

According to Viega and McGraw [28] "Reliability problems aren't always security problems, though we should note that reliability problems are security problems a lot more often than one might think [28]." Therefore, when security efforts are performed on a software system, some focus toward failure-prone components may reveal that those components are also likely to be vulnerable. *The objective of this research is to create and evaluate a model that predicts which components are likely to contain security faults based on non-security failures.*

[1] A component is one of the parts that make up a system [15].

We analyzed pre- and post-release non-security failure data and pre- and post-release security fault and failure data of a Cisco software system[2] to determine if non-security problems are associated with security problems. Since security faults are typically far fewer in number than non-security faults [1], not all of the failure-prone components will be associated with security faults. To be useful, the statistical model will have to determine which of the failure-prone components are also likely to be vulnerable. We constructed a model where the input variables to the model are non-security failures and the output of the model is a probability that a component in the system has at least one security fault.

The remainder of this paper is organized as follows. In Section 2 we provide background and related work, in Section 3 we detail the industrial case study, in Section 4 we present results, in Section 5 we present the limitations of the study, in Section 6 we provide a discussion, and finally in Section 7 we summarize and provide future work.

2 Background

In this section, we provide definitions of terms used throughout the paper. Seminal sources are used for each definition where possible. We also include prior work that compares security with reliability.

2.1 Definitions

External metrics - "Those metrics that represent the external perspective of software quality when the software is in use…These measures apply in both the testing and operation phases." [14]

Internal metrics - "Those metrics that measure internal attributes of the software related to design and code. These "early" measures are used as indicators to predict what can be expected once the system is in test and operation" [14].

Fault - "An incorrect step, process, or data definition in a computer program. Note: A fault, if encountered, may cause a failure" [15].

Fault-prone component - "A component that will likely contain faults" [4].

Failure – "The inability of a software system or component to perform its required functions within a specified performance requirements [15]."

Failure-prone component – A component that will likely fail due to the execution of faults [26].

Vulnerability - An instance of a [fault] in the specification, development, or configuration of software such that its execution can violate an [implicit or explicit] security policy [17].

[2] Due to the sensitivity of the security-related data, details of the system and data are omitted.

Vulnerability-prone component - A component that is likely to contain one or more vulnerabilities that may or may not be exploitable [8].

Attack - The inability of a system or component to perform functions without violating an implicit or explicit security policy. We borrow from the ISO/IEC 24765 [15] definition of failure to define attack, but remove the word "required" because attacks can result from functionality that was not stated in the specification.

Attack-prone component - A component that will likely be exploited [8].

An attack-prone component is a component that is likely to be exploited due to the types of vulnerabilities in that component. For example, the vulnerabilities may be easy to find, easy to exploit, or lead to desirable assets. A vulnerability-prone component that is not also attack-prone may contain vulnerabilities that are not easily found, are difficult to exploit, or do not lead to desirable assets. These characteristics represent our initial views of vulnerability- and attack-prone components [12].

In our setting, an attack is the execution of a security fault (vulnerability) that leads to a security failure. We use the context of execution to be consistent with the definitions and distinction between a fault and failure in the general reliability (non-security) context under the assumption that security is, by definition, a subset of reliability. System execution occurs during testing, internal usage, and in the field. In the context of testing, if a tester discovers a buffer overflow, then we say they have attacked the system. Although the tester may not have gone through the trouble of completely exploiting the buffer overflow to cause a denial-of-service or to inject code that escalates their privileges, the failure is a proof of concept that the system can be attacked. A risk value can be assigned to the security fault to describe how detrimental the vulnerability is to the system. Of course, static fault detection techniques can identify vulnerabilities that can be exploited, too.

2.2 Prior Work

The first and third authors performed two case studies on two different large commercial[3] telecommunications systems. The correlation between non-security and security failures was examined [9, 12]. We found a 0.8 ($p<.0001$) Spearman rank correlation between non-security system/feature failures and security failures for the first system and a 0.7 ($p<.0001$) correlation for the second system. For these two case studies the only available data were system/feature testing failures. The high correlations suggest that non-security failures are a good indicator of security problems and that security fortification efforts should be placed in the same areas of the software as reliability efforts. The case study presented in this paper attempts to replicate our previous studies on a different software system from a different vendor to determine if the statistical model yields consistent results.

Research papers comparing security failure data to non-security data are showing that reliability and security models are not dissimilar. Alhazmi et al. [1] compared the cumulative number of vulnerabilities for five different operating systems and found

[3] Due to the sensitivity of the data, the name of the vendor is omitted.

that the plots are analogous to reliability growth plots using logistic and linear models. Mullen et al. [18] have found the occurrence rate of security vulnerabilities follows the Discrete Lognormal distribution, which has also been shown in prior reliability growth, test coverage, defect failure rate, and code execution rates. Condon et al. [3] have found that security incident data can be modeled with Non-Homogenous Poisson Process models as done with reliability failure data. Lastly, Ozment and Schechter [23] found that Musa's Logarithmic model fit their OpenBSD security dataset to predict time-between-security-failures. We continue the examination of potential parallels between non-security and security problems by investigating if the *location* of security faults and failures can be approximated using non-security failure data.

2.3 Vulnerability- and Attack-prone Component Predictions

Neuhaus et al. [21] have also investigated predictive models that identify vulnerability-prone components. They created a software tool, Vulture, that mines a bug database for data including libraries and APIs which components are likely vulnerable. They performed an analysis with Vulture on Bugzilla, the bug database for the Mozilla browser, using imports and function calls as predictors. They were able to identify 45% of all of the vulnerable components in Mozilla. Shin and Williams [27] found a weak correlation (0.2) between complexity and security vulnerabilities in Mozilla, indicating that complexity contributes to security problems, but is not the only factor. We also found a 0.2 correlation between file coupling and vulnerability counts in a large telecommunications system [10]. In that case study, we used a classification and regression trees (CART, as discussed in Section 2.4) model to assign a probability of attack to each file. Upon ranking these probabilities in descending order, we found that 72% of the attack-prone files are in the top 10% of the ranked files and 90% are in the top 20% of the files. The input variables for that study consisted of the count of Klocwork[4] static analysis tools warnings, measure of file coupling, and count of added and changed source lines of code. In our other earlier work [11] we used a CART to predict which components were attack-prone using warnings from the static analysis tool, FlexeLint, and code churn. The model identified all of the attack-prone components, but with an 8% false positive rate. The study in this paper is based on a different type of system than our earlier studies.

2.4 Classification and Regression Trees (CART)

Our predictive model is comprised of a statistical technique and the independent variable non-security failure count. CART is a statistical technique that recursively partitions data according to X and Y values. The result of the partitioning is a tree of groups where the X values of each group best predicts a Y value. The leaves of the tree are determined by the largest likelihood-ratio chi-square statistic. The threshold or split between leaves is chosen by maximizing the difference in the responses between the two leaves [25]. For the case study in this paper, the X values are values

[4] http://www.klocwork.com/

from the non-security failures and the Y value is a binary value describing a component as attack-prone or not attack-prone. The CART technique has been shown to be useful for distinguishing failure-prone from not failure-prone components in the reliability realm [29].

3 Cisco Case Study

We analyzed pre- and post-release non-security failure data and pre- and post-release security fault and failure data that were submitted to the Cisco fault-tracking database in 2007 for a typical Cisco software system. The software system was divided into clearly defined components against which failures were reported. Each component consists of multiple files. The count of components was large enough to perform rigorous statistical analyses.

3.1 Non-security External Metrics as Predictors of Security Faults and Failures

The non-security failure reports were obtained from the Cisco fault-tracking database. The reports we used in our study included all severity 1, 2, and 3 non-security failure reports for the software system, where severity 1 is the highest impact to the customer. Most severity 1, 2, and 3 records in the fault-tracking database indicated actual problems in the software. Records with higher severity numbers had a stronger chance of being a feature request. During our failure report analysis, we eliminated duplicate failure records that represented a failure already reported in the system.

The non-security failure reports include failures observed during unit testing, function testing, performance testing, system testing, stress testing, alpha testing, beta testing, automated regression, internal use failures, early field trials, and customer-reported failures. Alpha testing is conducted on a production network within Cisco while beta testing is conducted on the customer site. The number and types of non-security failure reports are not disclosed for confidentiality reasons.

3.2 Security Fault and Failure Data

The security faults and failures are the dependent variables in this industrial case study. We included security faults and failures of all severity levels. We chose to study security faults and failures reported for the duration of a year (2007) to strengthen the goodness-of-fit of the predictive model we will build. Security faults and failures are rare events and difficult to model with small sample sizes. The number and types of security fault and failure reports are not disclosed for confidentiality reasons.

The security faults and failures were provided by the Cisco Security Evaluation Office that handles security data. The security faults in our study were reported during *static* inspections that were performed during the design and development stages of the software life cycle (SLC). The security failures were identified during

system *execution* and included problems from pre- and post-release testing and also include those security failures reported in the field.

In our setting, an attack-prone component is a component that contains at least one security fault or a security failure. We use the term attack-prone component instead of vulnerability-prone because most security faults were identified during system *execution*. Additionally, the attack-prone components had at least one security failure identified during pre- or post-release execution. We use the threshold of one security failure because there is little variability in the failure count per component and only one attack is needed to cause substantial business loss. Although some security failures were reported by customers, there was no evidence of successful attacks against the software. A component with no reported security faults or failures will be called a not attack-prone component in this paper.

4 Results

The analysis of the failure reports indicated that only a small percentage of the components consists of at least one security fault or failures. According to Pareto's law, 80% of the outcomes will be derived from 20% of the activities [6]. Although, this observation was originally described in the context of economics, it has also been observed in the context of faults in a software system [22]. The application of the law to the software setting is that software problems will not be evenly distributed across the software system. For example, in a survey of multiple software systems, it was shown that between 60% and 90% of software faults are due to 20% of the modules [2]. We observed Pareto's Law in our setting (see Section 4.2) because the distribution of attacks among the components is not evenly distributed across all components. All results in the sections below are reported on a per component basis.

The first analysis in our case study was to perform correlations between all of our non-security failures types (listed in Section 3.1) and counts of security faults and failures. The correlations with the highest coefficients will aid in independent variable selection during the construction of the model. Our statistical model will be a discriminatory model that classifies a component as attack-prone or not attack-prone. Associated with the classification is a probability of the component being attack-prone. In the event that the models cannot successfully discriminate between attack-prone and not attack-prone components, the correlations may indicate that a statistical technique does not perform well for the given dataset. For example, if we observe a high correlation between non-security failures and security faults and failures, but the discriminatory statistical approach cannot discriminate between attack-prone and not attack-prone components, then we would try a different statistical technique.

4.1 Correlations

We calculated[5] Spearman rank correlations to determine if an increase of non-security failures in a component is followed by an increase in count of security faults and failures for that component. The highest correlation, 0.4 ($p<.0001$), occurred between customer-reported non-security failures and the sum of security faults and failures as shown in Table 1. The correlations to the security fault and failure counts are low, but they are significant and represent that they have value for indicating the existence of security problems in a statistical model.

Table 1. Correlations between the non-security failures and vulnerabilities.

Count of non-security failures	Spearman rank correlation coefficient (p-value)
customer-reported	0.4 ($p<.0001$)
alpha testing	0.3 ($p<.0001$)
total non-security	0.3 ($p<.0001$)
internal use	0.3 ($p<.0001$)
system testing	0.2 ($p<.0001$)
performance testing	0.1 ($p<.0001$)
stress testing	0.1 ($p<.0001$)
beta testing	0.1 ($p<.0001$)
function testing	0.1 ($p=.04$)
early field trial testing	0.1 ($p<.0005$)

4.2 Classification of System Components

We performed classification analyses to discriminate between attack-prone components and not attack-prone components based on non-security failures. We built models using the discriminant analysis, logistic regression, and CART with the non-security failures enumerated in Section 3.1 as input variables. CART showed better separation between attack-prone and not attack-prone components than discriminant analysis and logistic regression. The non-security failure types that had the most predictive power in the CART model were alpha and beta testing non-security failures and customer-reported non-security failures.

The CART analysis splits the all of the system components into like groups based on the count of non-security alpha and beta testing failures and customer-reported non-security failures. The splits made in CART are shown in Appendix A. The values of *w*, *x*, *y*, and *z* are integer values and are not provided for confidentiality reasons.

In our analysis, the vulnerabilities were most likely to be in components where there are more than *x* customer-reported failures as denoted by the first (top most) split in the tree (see Appendix A). Failures from alpha and beta testing contributed less to the isolation analysis. The other failure types enumerated in Section 3.1 could not split the leaves to achieve separation between attack-prone and not attack-prone components as well as alpha and beta testing failures and customer-reported failures.

[5] All statistical analyses performed on SAS JMP 7.0.1.

The predictive power of the metrics is measured by the likelihood-ratio chi-square, G^2, of each input variable. A larger G^2 value indicates a more optimal split of a leaf in the CART analysis between attack-prone and not attack-prone components. In our model, the customer-reported problems contributed the most fit or separation in the overall model as shown in Table 2.

Table 2. Contribution of metrics to the model.

Non-security failure count	**Number of splits in tree**	**G^2**
customer-reported	3	205.7
alpha testing	1	7.4
beta testing	1	4.7
Total	5	217.8

We tested the input variables of the CART model to test for collinearity. Collinearity is defined as a high degree of correlation between the independent variables of a statistical model [7]. Collinearity occurs when an excessive number of input variables are used to determine an outcome, and the input variables measure the same outcome [7]. The highest correlation between our input variables, 0.2, existed between alpha testing and customer-reported failures and is a low correlation. The low correlations shown in Table 3 indicate that the failures identified by alpha and beta testing and customer-reported failures are measuring different types of failures or failures in different locations in the software. We included these input variables in the model because the correlations between them are low and thus the collinearity among them is small.

Table 3. Tests for collinearity in the independent variables.

Failure type	**Failure type**	**Spearman rank correlation coefficient**
alpha testing	customer-reported	0.2 ($p<.0001$)
beta testing	alpha testing	0.1 ($p<.0001$)
customer-reported	beta testing	0.1 ($p<.0001$)

The probability of an attack-prone component for a given leaf is given in Table 4. All components in a leaf have the same probability of being attack-prone. For example, in Leaf 1 100% of the components are attack-prone. In this leaf, all components should undergo security analyses. In Leaf 2 64% of the components are attack-prone and all components should go under security analyses, but 36% of the components will either not contain security faults or failures or contain security faults that are difficult to exploit or uninteresting to an attacker. The 36% false positive rate represents that time and effort spent on some components will not contribute greatly to the overall security posture of the software system. As shown in Table 4, there is a general likelihood ranking of attack-prone components. The components in Leaf 1 have the highest rank (probability of being attack-prone) and the components in Leaf 6 have the lowest rank.

In examining the efficacy of the model, the security efforts should be focused to all of the components in the first four leaves of the tree because the true positive rate (probability of finding an attack-prone component) is relatively high. In Leaves 5 and 6, the probability of identifying an attack-prone component is only ten percent and

one percent, respectively, representing that most security efforts would be wasted on low risk components. If we accept that the components in Leaves 1-4 are all attack-prone, then the model will have isolated 57% of the attack-prone components in the top nine percent (components in the top nine percent of the leaf-based ranking) of the system component ranking. Leaves 1-4 have a 48% Type I error (false positive) rate where not attack-prone components are interpreted by the model as attack-prone components. The remaining 43% of the vulnerable components are in Leaves 5 and 6 and represent the Type II error (false negative) rate. These attack-prone components would escape security efforts because they are in large groups of components with no reported vulnerabilities. Security engineers would not likely accept a scenario where most of their analyses are spent on components with low risk.

Table 4. Attack-prone probabilities in the leaves of the tree. The non-shaded rows represent the total system components in the top nine percent of the probability ranking. "Customer" represents the count of customer reported problems. "Alpha" represents the count of alpha testing failures. "Beta" represents the count of beta testing failures.

Leaf Number	Leaf Label	Probability not attack-prone	Probability attack-prone
1	customer=x&alpha>=y&beta>=y	0.00	1.00
2	customer>=x&alpha>=1&beta<y	0.36	0.64
3	customer>=x&alpha<1&customer>=z	0.27	0.73
4	customer>=x&alpha<1&customer<z	0.68	0.32
5	customer<x&customer>=w	0.90	0.10
6	customer<x&customer<w	0.99	0.01

The goodness-of-fit of the model can be determined by the receiver operating characteristic (ROC) curve. The ROC graph has the true positive rate on the y-axis and the Type I error rate on the x axis. The larger the area under the ROC curve, the better the goodness-of-fit. In Figure 1, the ROC curve for our model, represented by the thick line, has 88% of the area under the curve indicating that the non-security failures are a good metric for predicting which components are attack-prone. The thin line is a reflection of the solid curve to show the model's ability to classify not attack-prone components. The diagonal line represents the efficacy of the model if the predicted outcomes correctly identified 50% of the attack-prone components.

The R^2 value for the overall model is 36% indicating that not all of the variation in the data can be accounted for by CART. To validate the model, we performed five-fold cross-validation. Five has been shown to be a good value for performing cross-validation [13]. The 36% R^2 value we observed was based on the entire dataset. The cross-validation technique validates the R^2 value by testing the model on data the model has not used before to determine if the model is still effective [30]. The five-fold cross-validation divides ("folds") the total system components into five groups consisting of an approximately equal number of randomly chosen components. One group is used as the test set and the training set consists of the remaining four groups of components. The model is trained on the training set and the analysis is compared to the outcomes of the test set to validate how well the model performs on data that has not been "seen" before by the model. Each of the five groups of components has

one turn to be the test set which requires five analyses. After the five analyses are performed, the average error is calculated over the five trials. The cross-validated R^2 value was 34% indicating that the overall model is consistent with the model produced with the entire dataset. Despite the low R^2 values, all of the splits in the CART analysis were performed at or below the .05 significance level representing that each leaf split (separation between attack-prone and not attack-prone components) is statistically significant.

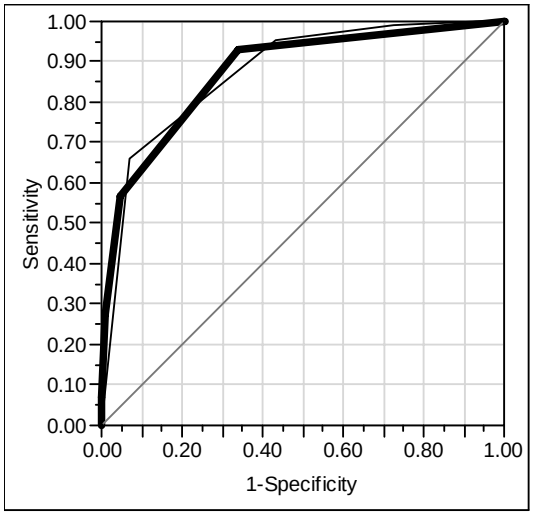

Figure 1. The ROC curve of the CART analysis.

5 Limitations

We cannot claim to have identified all faults in the software based on the failures that have surfaced during testing [5]. Additionally, the customer reported failures do not complete the identification of all non-security faults as predictors or all security faults and failures as dependent variables. Moreover, the testing effort may not have been equal for all components and thus components with fewer failures may appear more reliable or secure. Therefore, our analyses are based on incomplete data. The Type I (48%) and Type II (43%) error rates are high indicating that the model is not precise which, if applied at Cisco, could lead to effort wasted on low security risk components while some attack-prone components are never found. Additional metrics in a statistical model may help identify attack-prone components with lower Type I and Type II error rates. Furthermore, there are few security data making statistical analyses difficult. Lastly, the model presented in this paper is representative of one industrial software system and will not necessarily yield the same results on different software systems.

6 Discussion

In a Mozilla case study [27] and our earlier telecommunications system case study [11] the analyses showed that there is only a 0.2 correlation between complexity measures and security faults and failures. We observed that the 0.4 correlation between non-security failures and security failures in the software system is higher than complexity-related correlations. Further analysis is required to determine how complexity metrics correlate to security faults and failures in the system we studied. The higher the correlation, the more the predictor can contribute in a predictive model. While 0.4 correlation is low, it is significant, as are the complexity correlations. Combining these predictors into one model may build a useful model that has lower Type I and Type II error rates than a model with just one predictor.

According to Table 4, 57% of all attack-prone components were associated with greater than x (a value determined by the CART model) customer-reported non-security failures. Furthermore, Table 2 indicates that customer-reported non-security failures have the most ability to split components into groups of attack-prone and not attack-prone components. We provide three possible explanations for this observation. First, these observations indicate that the customer's operational profile[6] (usage) influences vulnerable execution paths not identified the testing techniques listed in Section 3.1. The more execution in those required features could increase the chance of a deviant operation profile that opens a security hole for an attacker.

The second possible reason why attack-prone components are associated with customer-reported failures is that they are the components that are most important (i.e. the reason the software was built) to the customer and thus those with the largest business risk. Therefore, failures with these components would more likely be considered security problems because they can be exploited by attackers to interrupt the software functions required by the customer. Failures in components that have less importance to a customer may be less likely to be a security problem because the impact of the failure does not preclude the customer from performing important tasks. However, the data indicate that 43% of the attack-prone components are associated with components with fewer than x customer-reported failures and show that the system can be exploited via components that are not as frequently as the others. Given the 57% and 43% percentages, security efforts should prioritize against the features that the customers use the most, but not exclude those components that are used less by the customers as they can also impact the customer.

Thirdly, the 0.4 correlation (a weak, but significant correlation) between non-security failures and security faults and failures may indicate that components with the most customer-reported failures are associated with deficiencies in the software process that lead to less reliable code. Gaps in the software process can lead to either the injection of a fault or the failure to remove a fault. The correlation between non-security failures and security faults and failures may indicate that the gaps in the software process lead to both reliability and security failures. The less failure-prone components (i.e. those with fewer or no security faults) indicate that the development groups with a stricter software process mitigate non-security problems at the same

[6] The complete set of operations (major system logical tasks) with their probabilities of occurrence [19].

time as security problems, perhaps without realizing that some of the risks they encountered were security-related. For example, if architectural risk analyses are not performed during design, then design flaws may not found until late in the software process when it is too late to change the design of the system. The design flaws may lead to unreliable functionality or a vulnerability that an attacker can exploit.

In our earlier work [11][7], we observed a 0.4 correlation between static analysis tool warnings and vulnerabilities found during testing and in the field. We did not observe a correlation between code churn and vulnerability counts. In our other work [10], we observed a 0.2 correlation between static analysis tool warnings and vulnerability counts. We also observed a 0.4 correlation between code churn and vulnerability counts. The vulnerability counts in these two datasets were small and thus may have hindered the identification of a stronger correlation between the predictors and vulnerability counts. Correlations between the same predictors and non-security faults/failures have been reported to be much stronger than the measurements presented in this paper. For example, Zheng et al. [31] observed a 0.73 correlation between static analysis tool warnings and testing and customer failures. Nagappan and Ball [20] observed correlations as high as 0.883 between code churn measures and general reliability defects/KLOC. If the static analysis tool warnings and code churn are strongly correlated to non-security problems and non-security problems are correlated to security problems, then static analysis tool warnings and code churn in our earlier work ([10, 11]) may have a stronger impact on security problems than what the correlations indicate. If true, the extensive research on reliability statistical models (e.g. [16, 22]) that have been shown to predict fault- and failure-prone components early in the SLC may also be helpful for security prediction models. The models can be modified to isolate security problems, or if we assume that the security faults cluster with the non-security faults, then security engineers can focus their efforts to the components predicted to be the most failure-prone by the reliability-based prediction model.

7 Summary and Future Work

We analyzed a Cisco software system to determine if non-security problems are associated with security problems. We found a 0.4 correlation between security faults and failures and non-security failures suggesting that general reliability of a software component is an indicator of the security posture of that component. Our CART model shows that alpha and beta testing failures and customer-reported failures can discriminate between attack-prone and not attack-prone components. Additionally, the model provides a threshold of non-security failure counts “required” to have a security fault or failure. This threshold is useful for determining which of the failure-prone components should receive security attention in application of the idea suggested by Viega and McGraw [28] in the Introduction. The model correctly identified 57% of the attack-prone components in the top nine percent of the

[7] For the detailed version of this paper, see: M. Gegick, L. Williams, and J. Osborne, "Predicting Attack-prone Components with Internal Metrics," NC State University, Raleigh, TR-2008-08, 25 February 2008.

components when ranked by probability of being attack-prone. The CART analysis showed that the best indicator of security faults and failures were those components with the most customer-reported failures. This observation suggests that the customers' operational profiles may influence vulnerable execution flows in the software that were not identified during pre- or post-release testing. We conclude that non-security failures and the predictors of non-security failures are potential metrics security-related predictive models. Given that reliability and security problems exist in the same locations, it may be worthwhile to unify the concepts of "software reliability engineering" and "software security engineering" into a single theme (e.g. *software assurance engineering*) to indicate that security and reliability folks should collaborate in the same sections of the software system and that security should be kept in mind when the system becomes unreliable. Next, we will examine if the non-security failures in our dataset can actually afford an attacker a means to exploit the software system. The initially classified non-security failures may provide opportunities for new types of security attacks. These security holes may result from not abiding by the principle of fail-safe defaults because the failure to perform the required functions within the specified performance requirements opened a security hole. Fail-safe defaults is a security design principle [24].

Acknowledgment

This work is supported by the National Science Foundation under CAREER Grant No. 0346903. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] O. H. Alhazmi, 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] B. Boehm and V. Basili, "Software Defect Reduction Top 10 List," IEEE Computer, vol. 34, no. 1, pp. 135-137, January, 2001.

[3] E. Codon, M. Cukier, and T. He, "Applying Software Reliability Models on Security Incidents," *International Symposium on Software Reliability Engineering*, Trollhattan, Sweden, 2007.

[4] G. Denaro, "Estimating software fault-proneness for tuning testing activities," *International Conference on Software Engineering*, St. Malo, France, pp. 269-280, 2000.

[5] E. Dijkstra, *Structured Programming*, Brussels, Belgium, 1970.

[6] A. Endres and R. D. Rombach, *A Handbook of Software and Systems Engineering*, Harlow, England, Pearson Education, Limited, 2003.

[7] R. Freund, R. Littell, and L. Creighton, *Regression Using JMP*, Cary, NC, SAS Institute, Inc., 2003.

[8] M. Gegick and L. Williams, "Toward the Use of Static Analysis Alerts for Early Identification of Vulnerability- and Attack-prone Components," *First International Workshop on Systems Vulnerabilities (SYVUL '07)* Santa Clara, CA, July 1-6 2007.

[9] M. Gegick, "Failure-prone Components are also Attack-prone Components," *OOPSLA - ACM student research competition*, Nashville, Tennessee, pp. 917-918, October 2008.
[10] M. Gegick and L. Williams, "STUDENT PAPER: Ranking Attack-prone Components with a Predictive Model," *International Symposium on Software Reliability Engineering*, Redmond, WA, pp. 315-316, November 2008.
[11] M. Gegick, L. Williams, J. Osborne, and M. Vouk, "Prioritizing Software Security Fortification through Code-Level Security Metrics," *Workshop on Quality of Protection*, Alexandria, VA, pp. 31-37, 2008.
[12] M. Gegick, L. Williams, and M. Vouk, "Predictive Models for Identifying Software Components Prone to Failure During Security Attacks," Build Security In (https://buildsecurityin.us-cert.gov/daisy/bsi/home.html) 2008.
[13] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning*, New York, Springer, 2001.
[14] ISO, "ISO/IEC DIS 14598-1 Information Technology - Software Product Evaluation - Part 1: General Overview," October 28 1996.
[15] ISO/IEC 24765, "Software and Systems Engineering Vocabulary," 2006.
[16] T. M. Khoshgoftaar, E. B. Allen, A. Naik, W. Jones, and J. P. Hudepohl, "Using Classification Trees for Software Quality Models: Lessons Learned," *International Journal on Software Engineering and Knowledge Engineering*, vol. 9, no. 2, pp. 212-231, 1999.
[17] I. Krsul, "Software Vulnerability Analysis," PhD Thesis in Computer Science at Purdue University, West Lafayette 1998.
[18] R. Mullen and S. Gokhale, "A Discrete Lognormal Model for Software Defects Affecting QoP," *Quality of Protection*, Milan, Italy, 15 September 2005.
[19] J. D. Musa, *Software reliability engineering: More reliable software faster and cheaper Second Ed.*, Bloomington, Indiana, AuthorHouse, 2004.
[20] N. Nagappan and T. Ball, "Use of Relative Code Churn Measures to Predict Defect Density," *International Conference on Software Engineering*, St. Louis, MO, pp. 284-292, 15-21 May 2005.
[21] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting Vulnerable Software Components," *Computer and Communications Security*, Alexandria, VA, pp. 529-540, 29 October-2 November 2007.
[22] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Where the bugs are," *International Symposium on Software Testing and Analysis*, Boston, Massachusetts, pp. 86-96, 2004.
[23] A. Ozment and S. Schechter, "Milk or wine: does software security improve with age?," *15th Conference on USENIX Security Symposium*, pp. 93-104, July 2006.
[24] J. Saltzer and M. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278-1308, September 1975.
[25] SAS Institute Inc., "The Partition Platform," SAS Institute, Inc., Cary, NC, 2003.
[26] A. Schroter, T. Zimmermann, and A. Zeller, "Predicting Component Failures at Design Time," *International Symposium on Empirical Software Engineering*, Rio de Janeiro, Brazil, pp. 18-27, September 21-22 2006.
[27] Y. Shin and L. Williams, "Is Complexity Really the Enemy of Software Security?," *Workshop on Quality of Protection*, Alexandria, VA, pp. 47-50, 2008.
[28] J. Viega and G. McGraw, *Building Secure Software How to Avoid Security Problems the Right Way*, Boston, Addison-Wesley, 2002.
[29] M. Vouk and K. C. Tai, "Some Issues in Multi-Phase Software Reliability Modeling," *Center for Advanced Studies Conference (CASCON)*, Toronto, pp. 512-523, October 1993.
[30] I. Witten and E. Frank, *Data Mining*, Second ed. San Francisco, Elsevier, 2005.
[31] J. Zheng, L. Williams, W. Snipes, N. Nagappan, J. Hudepohl, and M. Vouk, "On the Value of Static Analysis Tools for Fault Detection," *IEEE Transactions on Software Engineering*, vol. 32, no. 4, pp. 240-253, April 2006.

Appendix A: Classification and Regression Tree

Non-shaded boxes represent leaves of the tree.