Beyond Heuristics: Learning to Classify Vulnerabilities and Predict Exploits

Mehran Bozorgi, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker
Department of Computer Science and Engineering
University of California, San Diego
mehranbozorgi@google.com, {saul,savage,voelker}@cs.ucsd.edu

ABSTRACT

The security demands on modern system administration are enormous and getting worse. Chief among these demands, administrators must monitor the continual ongoing disclosure of software vulnerabilities that have the potential to compromise their systems in some way. Such vulnerabilities include buffer overflow errors, improperly validated inputs, and other unanticipated attack modalities. In 2008, over 7,400 new vulnerabilities were disclosed—well over 100 per week. While no enterprise is affected by all of these disclosures, administrators commonly face many outstanding vulnerabilities across the software systems they manage. Vulnerabilities can be addressed by patches, reconfigurations, and other workarounds; however, these actions may incur down-time or unforeseen side-effects. Thus, a key question for systems administrators is which vulnerabilities to prioritize. From publicly available databases that document past vulnerabilities, we show how to train classifiers that predict whether and how soon a vulnerability is likely to be exploited. As input, our classifiers operate on high dimensional feature vectors that we extract from the text fields, time stamps, cross-references, and other entries in existing vulnerability disclosure reports. Compared to current industry-standard heuristics based on expert knowledge and static formulas, our classifiers predict much more accurately whether and how soon individual vulnerabilities are likely to be exploited.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; I.5.1 [**Pattern Recognition**]: Models—*Statistical*; I.5.2 [**Pattern Recognition**]: Design Methodology—*Feature evaluation and selection*

General Terms

Algorithms, Security

Keywords

supervised learning, SVM, vulnerabilities, exploits

KDD'10, July 25–28, 2010, Washington, DC, USA.

1. INTRODUCTION

Among the many requests made of researchers in computer security, few are as frequent or as urgent as the call for meaningful security metrics. The requests are driven by a widespread need to quantify security risks ("how likely is it that an attacker will thwart my security measures?") in a way that informs operational policy choices. Unfortunately, the adversarial nature of security has resisted traditional methods of quantifying risk and has even led some to argue that such metrics are inherently unattainable [4]. Nevertheless, even absent a comprehensive solution to this conundrum, there remains a need to evaluate the value of distinct operational security choices. Thus, a range of ad hoc approaches have emerged in individual domains where the needs are particularly acute. In this paper we focus on one such domain — evaluating vulnerability disclosures — and we show that it is possible to make meaningful predictions using tools from data mining and machine learning.

Public vulnerability disclosure has long been a staple of the software security industry, with many thousands of new software vulnerabilities identified and publicized each year [11]. In turn, these vulnerabilities are communicated, via a variety of channels, to system administrators who must then determine if they have susceptible systems and decide what action to take if so. Unfortunately, patching and other mitigations can incur significant manpower overheads (even more so for mission critical services that require quality assurance testing before deploying new software). Since few organizations have the resources to address every vulnerability disclosure that might impact their enterprise, administrators must prioritize their efforts, triaging the most critical vulnerabilities to address first.

To inform these decisions, a variety of "vulnerability scoring" frameworks have been designed to assess risk qualitatively (e.g., Microsoft's critical, important, moderate, or low severity rating) or quantitatively (e.g., US-CERT's severity metric). Indeed, one such framework — FIRST's Common Vulnerability Scoring System (CVSS) [9] — appears to be emerging as the *de facto* standard in the community. The use of CVSS is mandated in the Payment Card Industry's Data Security Standard (PCI-DSS), it is the ranking used in NIST's National Vulnerability Database (NVD), and its use is recommended by a wide range of computer, networking and software vendors (e.g., Cisco's Risk Triage whitepaper [5]).

However, while these systems are carefully designed using expert knowledge, they are inherently ad hoc in nature. For example, the CVSS (v2) overall "Base Score" is expressed in terms of Impact (I) and Exploitability (E) components by:

$$\text{BaseScore} = (1.176) * \left(\frac{3I}{5} + \frac{2E}{5} - \frac{3}{2}\right). \qquad (1)$$

By convention, this score is rounded to one decimal place, and it

is set to zero (regardless of the above formula) if $I = 0$. We note further that the Impact and Exploitability components in eq. (1) are themselves combinations of categorical magic numbers. (For example, the Exploitability component depends on an Access Vector score which takes the value 0.395 if the vulnerability requires local access, 0.646 if the vulnerability requires adjacent network access, and 1.0 if the vulnerability global network access). While we have little doubt that these scoring metrics were carefully considered and of great value when first developed, we suspect that any single fixed equation, such as eq. (1), is unlikely to provide a robust and lasting model of vulnerability severity.

To this end, our paper seeks to place this problem on a more systematic footing. Using tools from machine learning, we show how to train classifiers that predict whether vulnerabilities are likely to be exploited, and if so, how soon. Our results suggest that our trained classifiers are likely to outperform current measures of exploitability. In particular, our classifier outputs correlate much better with vulnerability outcomes than the "Exploitability" component of the CVSS Base Score in eq. (1).

2. BACKGROUND

Software vulnerabilities are exploitable flaws in software systems that pose significant security risks. Production software inevitably ships with many such flaws, a subset of which are subsequently discovered and become known over time. When flaws are discovered, vendors distribute patches and mitigations to their customers, who ideally implement such measures before an *exploit* is developed and targeted against them. This vulnerability lifecycle (described in more detail by Arbaugh *et al.* [1]) has in turn driven the creation of a complex ecosystem of players: vulnerability researchers, software and security vendors, security information providers and a range of networks, information sources and markets connecting them together (for a comprehensive analysis of the vulnerability ecosystem see Frei *et al.* [10]).

2.1 Public Vulnerability Disclosures

At the end of this process, vulnerabilities are documented and disclosed to the public. These reports not only list various discrete attributes of each vulnerability (e.g., software affected, date disclosed), they also describe (in plain text) how each vulnerability works, why it presents a threat, and how it can be mitigated. This information is disclosed to the public via multiple sources, including moderated forums (e.g., Bugtraq [21]), individual vendors (e.g., Microsoft [14], commercial aggregators (e.g., Secunia [20]), and open source databases (e.g., OSVDB [17]). Vulnerabilities are also quickly assigned a unique identifier — both local to individual information providers/repositories and global across multiple vulnerability databases using MITRE's Common Vulnerabilities and Exposures (CVE) service [6].

The precise timing of vulnerability disclosures depends considerably on how they were found and the policy of the organizations involved. Some vulnerabilities are disclosed immediately upon discovery while others may be kept private for significant periods of time to allow vendors to develop and test appropriate patches and mitigations. Still other vulnerabilities are exploited before or at the same time as public disclosure (so-called 0-day exploits). Vulnerability discovery and disclosure policy has generated a great deal of both controversy and research [2, 3, 18, 16]. A number of studies have also examined the probability that vulnerabilities are able to be patched [15, 19]. By contrast, our work focuses on predicting if a vulnerability is likely to be exploited shortly (thus meriting immediate attention from system administrators).

2.2 Rating Vulnerabilities

To aid system administrators, vulnerability disclosures typically include a qualitative or quantitative assessment of each vulnerability's severity. The overall severity score depends on both *impact* (how significant are the consequences of exploitation) and *exploitability* (how difficult is the vulnerability to exploit). Severity scores are derived primarily from expert knowledge and/or communal input. For example, US-CERT generates a quantitative severity score ranging from 0 to 180, calculated directly from answers to a range of qualitative questions (e.g., "Is information about the vulnerability widely available or known?" and "What are the preconditions required to exploit the vulnerability?") [7]. Microsoft's Security Bulletin documents vulnerability severity using a qualitative scheme (critical, important, moderate, or low) as do Secunia's reports (extremely critical, highly critical, moderately critical, less critical, or not critical). More recently, a group of vendors and researchers came together, under the sponsorship of the Forum of Incident Response and Security Teams (FIRST), to create a new severity metric, the Common Vulnerability Scoring System (CVSS). Now in its second iteration, CVSS defines several independent metrics, but it is the "base metric" which is typically used in third-party vulnerability databases. This score combines impact and exploitability components according to a carefully designed formula [9].

Unfortunately, each of these systems measures different things and weights them in different ways. We are unaware of any empirical study evaluating the effectiveness of any of these metrics or comparing them to one another. Thus, it is hard to make concrete statements about which approach is best, or why. Indeed, this problem is inherently difficult since some aspects of "severity" are either context dependent (e.g., a mission critical server being shut down may be more "severe" than a print server) or may be inherently difficult to quantify. However, the issue of exploitation is far more clear cut — a vulnerability is either exploited or not and the date upon which a working exploit becomes known is frequently documented. Thus, in this paper we focus on the exploitability aspect of severity.

Given this scope, we argue that existing scoring systems are probably too limited to offer strong predictive power. They include only a few factors in each vulnerability's assessment—which may not be the key distinguishing features and frequently depend on the judgment of evaluators—and they combine these features in the same way to produce a score for widely different sorts of vulnerabilities. For example, the current CVSS Exploitability score is calculated as follows [13]:

$$\text{Exploitability} = 20 * AV * AC * \text{Authentication} \quad (2)$$

where AV stands for `Access Vector` and AC stands for `Access Complexity`, and each of these variables are assigned particular fixed values based on other qualitative or subjective evaluations. For example, AC is set to 0.35 if access complexity is deemed to be "high", 0.61 if "medium" and 0.71 if "low". It is not entirely clear how this formula or its constants were designed. Moreover, it seems unlikely that this simple formula can model the probability of exploitation across many different sorts of vulnerabilities.

3. VULNERABILITY DATA

In this study, we use two well-known, online sources of vulnerability data, the Open Source Vulnerability Database (OSVDB) [17] and the MITRE Common Vulnerabilities and Exposures (CVE) database [6].

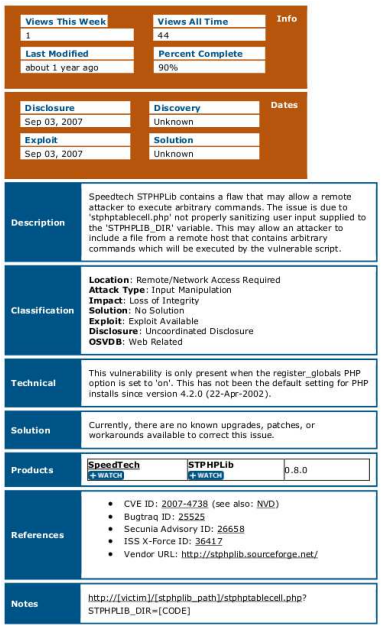


Figure 1: An example OSVDB vulnerability report.

Exploit Category	# Vulnerabilities	Label
Exploit Available	8,537	Positive
Exploit Rumored / Private	1,483	Positive
Exploit Unavailable	536	Negative
Exploit Unknown	3,209	Negative
No Category	999	Not Used
Total	14,764	

Table 1: Categories of exploited vulnerabilities.

OSVDB is a large database containing reports on over 57,000 vulnerabilities. As an example, figure 1 shows the OSVDB report for a vulnerability in a Web services library. These reports contain a wealth of information about each vulnerability, indexed using a unique OSVDB ID, including a detailed description, technical details, the software products affected, solutions (such as patches and mitigations to prevent exploitation), and references to other sources of information about the vulnerability. The reports also include classification information, such as the type of attack required to exploit the vulnerability, the origins from which an attack can be launched, and the manner in which the vulnerability was disclosed. Section 4.1 describes how we extract information from these reports as features for classification and prediction.

The reports also provide temporal information for each vulnerability, such as the dates of discovery, disclosure, and first known exploits. We augment this data with additional temporal information provided by Frei *et al.* as described in their WEIS 2009 paper [10]. As described below, we use this additional information to create labeled training and test sets.

From the OSVDB database we extracted a large set of vulnerabilities for classification and prediction. We used only vulnerabilities that were disclosed during the years 1991–2007, inclusive. Vulnerabilities before 1991 represent a different era of software; we excluded later vulnerabilities because, at the time we started the project, they were recent and still in flux (e.g., many of them had undetermined outcomes). We also excluded vulnerabilities that did not have a description.

Table 1 shows the number of vulnerabilities we use in our experiments and how we label them. It categorizes vulnerabilities using their OSVDB "Exploit Classification" status. If a vulnerability has an available, rumored, or private exploit, we label it as a "positive" vulnerability, indicating that it has been exploited. Similarly, if a vulnerability has no known exploits or exploits are unavailable, we label it a "negative" vulnerability indicating that it is not exploited. If a vulnerability report does not classify its exploit status, we exclude it from consideration since we cannot determine the accuracy of our predictions. Section 4.2 describes how we train a classifier from these labeled examples of vulnerabilities.

We use the CVE database to augment the vulnerability reports from the OSVDB database. Similar to OSVDB, CVE entries include summaries, references to related products and reports, information about the type of vulnerability, time stamps, and severity scores. In addition to providing more information that can be extracted as features for classification, for some vulnerabilities the CVE entries also provides information missing in the OSVDB reports. We integrate these records by cross-referencing their CVE and OSVDB identifiers. Most OSVDB reports reference the corresponding CVE reports for the same vulnerability and conversely, some CVE entries have corresponding OSVDB IDs as well.

Finally, we note that the quality of our results are inherently tied to the quality of this disclosure data and in particular the quality of the temporal labels (when a vulnerability was disclosed and exploited). This creates two potential classes of problems. In principle, there are adversarial training risks since bad vulnerability data could influence what the classifier learns during training. However, we believe this is a particularly unlikely scenario since vulnerability databases are generated by large numbers of independent actors. It seems unlikely that an adversary would discover and disclose enough new vulnerabilities (in turn validated and accepted by third parties) to influence the overall feature set used in training. Similarly, while an adversary might try to "game" our predictions (e.g., by only exploiting vulnerabilities which we had classified as unlikely to be exploited), the risk seems low, and certainly such a counterstrategy is no easier than it is under current vulnerability scoring systems. A somewhat more likely limitation is systematic bias. In particular, we note that large numbers of vulnerabilities in the complete database have unknown exploitation status or dates, which limits our ability to train on these records. In this work, we assume that the remainder of disclosures (with known status and dates) are representative and accurate. A selection bias would emerge if the omitted records were systematically different than complete records; however, we do not believe such a bias exists.

4. MACHINE LEARNING FOR VULNERABILITY CLASSIFICATION

We aim to improve on existing approaches by casting vulnerability classification as a problem in machine learning. In a nutshell, our goal is to replace small-scale heuristics by large-scale statistics. This section describes our statistical model for vulnerability classification. The model is estimated from a large database of vulnerabilities that have been labeled as "exploited" or "not exploited". Section 4.1 describes how we extract information from this database and distill it into feature vectors for classification, and section 4.2 describes how we classify these feature vectors using support vector machines [22]. The training and test sets of feature vectors in our experiments are available at `http://www.sysnet.ucsd.edu/projects/exploit-learn/`.

Feature Family	Count	Database Source
Summary (B)	14883	CVE
Full Product Name (B)	13040	OSV - Obj. Correls.
Description (B)	11573	OSV - Vulnerabilities
Title (B)	9812	OSV - Vulnerabilities
Short Description (B)	9761	OSV - Vulnerabilities
Manual Notes (B)	6576	OSV - Vulnerabilities
Product Versions (B)	5388	OSV - Obj. Versions
Related Products (B)	5057	CVE
Product Names (B)	3661	OSV - Obj. Products
Tech. Description (B)	3479	OSV - Vulnerabilities
Solution (B)	3474	OSV - Vulnerabilities
Product Vendors (B)	2500	OSV - Obj. Vendors
Authors (B)	2368	OSV - Credits
Keywords (B)	1556	OSV - Online
References (B)	267	CVE
Classifications (B)	69	CVE
External Refs (B)	31	OSV - Ext. Refs.
OSVDB Dates (I)	15	OSV - Vulnerabilities
Attack Type (B)	11	OSV - Classifications
Category (B)	9	OSV - Classifications
Location (B)	8	OSV - Classifications
Solution Category (B)	8	OSV - Classifications
Disclosure Type (B)	8	OSV - Classifications
CVE Dates (I)	6	CVE
Impact (B)	4	OSV - Classifications
Scores (I)	3	CVE
Effect on Products (B)	3	OSV - Aff. Types
Other (I)	8	OSV & CVE
Total	93578	

Table 2: Extracted features from the vulnerability data. (B) denotes binary and (I) denotes integer features.

4.1 Feature extraction

Our database of vulnerabilities contains a wealth of information, both factual and textual, about their histories and distinguishing characteristics. For each vulnerability, we extract a high-dimensional ($d = 93578$) feature vector of binary and integer-valued features. Though many of these features will ultimately turn out to be irrelevant or redundant for classification, the goal of our feature extraction is to distill as much information as possible for subsequent statistical analysis.

Much of our information about vulnerabilities is contained in text fields. We derive binary features using a bag-of-words representation for each text field [12]. Essentially, these features record whether or not particular tokens (e.g., “buffer", “heap", “DNS") appear in specific text fields (e.g., “title", “solution", “product name") associated with each vulnerability.

Table 2 shows the breakdown of features that we extract for each vulnerability in our database. Each row in the table indicates the number of features derived from a particular type of information. Most of the features are generated from bag-of-words representations of text fields. However, integer-valued features also encode useful information, such as the date when a vulnerability was first disclosed, the length of text describing its symptoms, or the ranking of its severity according to other popular heuristics.

4.2 Large margin classification

We build classifiers by training linear support vector machines (SVMs) [22] on the feature vectors described in the previous section. (As preprocessing, however, the non-binary features are normalized to lie between zero and one so that they do not overshadow the binary features.) Linear SVMs are trained by computing the maximum margin hyperplane that separates the positive and negative examples in feature space. The decision rule mapping feature vectors $\mathbf{x} \in \Re^d$ to labels $y \in \{-1, +1\}$ is given by:

$$y = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b), \tag{3}$$

where $\mathbf{w} \in \Re^d$ is the normal (weight) vector to the separating hyperplane and b is the distance of the separating hyperplane from the origin.

Linear SVMs are particularly appropriate for our application to vulnerability classification because we have many more input features (d) than training examples (n). In particular, for the experiments in section 5, the ratio of features to examples is never less than 10-to-1. In this regime of small sample size ($n \ll d$), there are many hyperplane decision boundaries that can perfectly separate all n examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ in our training sets. Linear SVMs compute the hyperplane that (roughly speaking) maximizes the distance of the most borderline training examples to the linear decision boundary. This hyperplane is not only uniquely specified, but a large body of work in statistical learning theory also shows that it generalizes better to new data, yielding lower expected error rates when used to classify previously unseen examples [22]. Many software packages are available for fitting models of this form; for the results in this paper, we used the LIBLINEAR implementation of SVMs [8].

5. EVALUATION

In this section we present our experimental results using SVMs for vulnerability classification. We consider several different scenarios. We first evaluate the prediction accuracy in an offline experiment, representing a best-case scenario where we consider the data set of vulnerabilities as a single, static snapshot; we also examine the features that have the most prominent role in making predictions. We then evaluate the prediction accuracy of our model in an online experiment emulating a real-world deployment: we dynamically update the classifier and make predictions over time as new vulnerabilities appear. We also use SVMs to predict if vulnerabilities will be exploited within a particular time frame. Finally, we compare the results from SVMs to current heuristic approaches to vulnerability classification.

5.1 Methodology

As discussed in Section 3, we use the OSVDB and CVE databases as our data set of vulnerability examples. In addition to providing the features we use for learning and classification, they also provide the ground truth for evaluating the accuracy of our classifiers. In general, we label those vulnerabilities that have exploits as positive examples and vulnerabilities that do not have exploits as negative examples. Table 1 shows a breakdown of the vulnerabilities based on these labels.

Note that there are more positive examples (10,020) than negative examples (3,745), i.e., more vulnerabilities with exploits than those without. When conducting balanced experiments, which remove any such bias in the data used for classification, we randomly choose the same number of examples from both sets multiple times and average the results. When conducting unbalanced experiments, an unavoidable aspect of practical deployments, we explicitly quantify and report the bias in the input data. Finally, a subset of the vulnerabilities do not have exploit information; we do not use these examples in our experiments because, without true labels, we cannot evaluate the accuracy of classification.

	Training	Testing
Positive Examples ($\|P\|$)	1600	2000
Negative Examples ($\|N\|$)	1500	1874
Total Examples	3100	3874
True Negatives	100%	92.2%
True Positives	100%	87.5%
False Negatives	0%	7.79%
False Positives	0%	12.5%
Total Accuracy	100%	89.8%

Table 3: Prediction accuracy in the offline experiment.

5.2 Offline Exploit Prediction

In our first experiment, we evaluate how well SVMs classify vulnerabilities in an offline setting. For this experiment, we use a balanced data set of roughly 4000 positive and negative examples—that is, divided almost evenly between vulnerabilities with and without exploits. We train SVMs on 40% of these examples (as training data) and evaluate them on 50% of these examples (as test data). We use the remaining 10% of examples as a development set to choose tuning parameters for the SVMs. We report averaged results from ten-fold cross validation: that is, we learn ten different classifiers, randomly choosing which examples fall into the training, test, and development sets, then average the results across all runs.

Table 3 shows the results. Here, *true positives* are positive examples correctly classified as vulnerabilities that will be exploited; *true negatives* are negative examples correctly classified as vulnerabilities with no known exploits; *false positives* are positive examples incorrectly classified as vulnerabilities that will be exploited, but were not; and *false negatives* are negative examples that were incorrectly classified as vulnerabilities that have no known exploits, but in fact do. The overall accuracy is nearly 90%, demonstrating the viability of statistical methods for vulnerability classification.

5.3 Feature Inspection

The offline classification results show that the vulnerability reports contain useful features for predicting whether a vulnerability will be exploited. We now examine which features play a prominent role in these predictions. In the linear SVMs that we use, the decision rule in eq. (3) multiplies each feature by a positive or negative weight. Recall that all features are normalized to lie between zero and one before the weights are learned. Thus the magnitudes of these weights reflect the relative contribution of each feature to the decision rule.

Tables 4 and 5 show the top 10 features with the highest positive and negative normalized weights, respectively, from the experiment in Section 5.2. Positively weighted features suggest to the classifier that the vulnerability has an exploit; negatively weighted features suggest to the classifier that the vulnerability does not. The first column lists the feature family (Table 2) and the second column the specific feature in the family. For example, the feature "References: BUGTRAQ ID" in the first row of Table 4 corresponds to the token "BUGTRAQ ID" appearing in the "References" feature family of a vulnerability report. The third column lists the number of vulnerabilities N_j in which feature j appears in the training data. For example, the feature "References: BUGTRAQ ID" occurs in 2,045 of the 3,100 vulnerabilities in the training set. The fourth column lists the *normalized* weight of the feature. The normalized weight $\tilde{w}_j = w_j(N_j/N)$ is the raw weight w_j learned by the classifier multiplied by the ratio of the number of vulnerabilities N_j whose jth feature is non-zero divided by the total number of vulnerabilities in the training set ($N = 3100$). By sorting the normalized weights, we reveal the features with the largest overall effect across all vulnerabilities (as opposed to the largest effect on a possibly miniscule number of vulnerabilities).

Feature Family	Feature	N_j	Weight
References	BUGTRAQ ID	2045	0.3674
Modified Date − Create Date	(Time Difference)	3096	0.1860
Authors	(Number of Tokens)	1858	0.1461
Class. Types	LOCATION	2509	0.1373
References	(Number of Tokens)	3054	0.1292
Title	(Number of Tokens)	3085	0.1229
CVE Summary	ALLOWS	1983	0.1146
Notes	(Number of Tokens)	925	0.0971
Modified Date − Disclosure Date	(Time Difference)	3098	0.0902
References	SECUNIA ADV. ID	2107	0.0840

Table 4: Top 10 features with the highest positive normalized weights, and the number of vulnerabilities N_j in which they appear in the training set. Features prefixed with "CVE" are derived from CVE entries, otherwise they come from OSVDB reports.

The features in Table 4 are those that suggest most strongly to the classifier that a vulnerability will be exploited. Many of them correspond to the number of tokens in particular feature families, such as the "Authors", "Title", and "Notes" sections of the vulnerability reports. These weights suggest that vulnerabilities with exploits generally have longer reports in the vulnerability databases than those without exploits; when looking at the vulnerability reports manually, we find that this situation is indeed the case. Other top features are references to other security databases, suggesting that vulnerabilities with exploits are often tracked by multiple sources. Finally, we note that there are many features with positive weights beyond those in Table 4. The top 100 features span nearly all of the feature families listed in Table 2; in other words, there are useful features in all parts of the vulnerability reports.

The features in Table 5 are those that suggest most strongly to the classifier that a vulnerability will not be exploited. Among these features, we observe two trends. First, we see that multiple features measuring the passage of time have strongly negative weights. Thus it appears that vulnerabilities with "dusty" reports are less likely to be exploited. Second, we see that vulnerabilities whose product-related fields are undefined ("Full Product Name", "Products", "Vendors") also appear less likely to be exploited. Presumably, such "incomplete" reports indicate vulnerabilities that have not received much attention from the community (nor from attackers).

5.4 Online Exploit Prediction

The offline experiment in Section 5.2 showed the potential for learning to classify vulnerabilities that were randomly divided into training, test, and development sets. In a real-world deployment, however, system administrators would train the classifier on known vulnerabilities to make predictions about new ones. Moreover, as time goes, the knowledge that vulnerabilities have or have not been exploited can be used to create new training examples. We can then extend the training set with these new vulnerabilities and learn a new classifier based on the most up-to-date information. This process can continue indefinitely as time progresses.

Feature Family	Feature	N_j	Weight
Today − Last Modified Date	(Time Difference)	3097	-0.9432
CVE Today − Generate Date	(Time Difference)	3045	-0.1478
Class. Types	ATTACK TYPE	3052	-0.1439
CVE Mod. Date − Generate Date	(Time Difference)	3044	-0.1412
Classifications	ATTACK TYPE INPUT MANIP	2158	-0.08260
References	RELATED OSVDB ID	1486	-0.07923
Create Date − Disclosure Date	(Time Difference)	3098	-0.06584
Description	CODE	872	-0.05499
CVE References	VUPEN	994	-0.04956
Full Product Name	(Not Defined)	2322	-0.04770
Products	(Not Defined)	2322	-0.04770
Vendors	(Not Defined)	2322	-0.04770

Table 5: Top 10 features (including ties) with the lowest negative normalized weights, and the number of vulnerabilities N_j in which they appear in the training set. Features prefixed with "CVE" are derived from CVE entries, otherwise they come from OSVDB reports.

In our next experiment we emulate this online scenario using the time-stamp information in our vulnerability databases:

1. Initialize a baseline classifier — Consider all vulnerabilities $\{V\}_0^t$ reported between time 0 and time t that have known outcomes (exploited or not) represented with labels $\{L\}_0^t$. We build a baseline classifier C_t based upon these vulnerability examples and labels $\{V, L\}_0^t$.

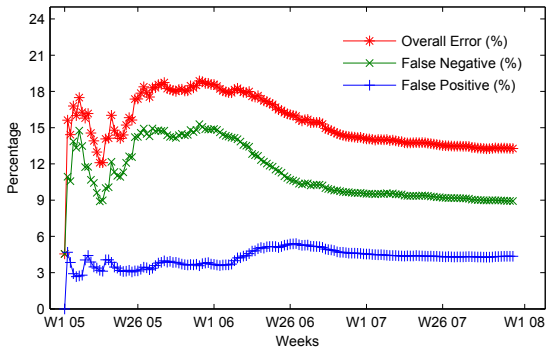
2. Predict exploited vulnerabilities that appear in the next time interval — Suppose new vulnerabilities $\{V\}_{t+1}^{t+T}$ arrive after a duration of T time elapses. We can use C_t to predict the labels $\{L'\}_{t+1}^{t+T}$ for these examples.

3. Update with known vulnerability outcomes — Once we know whether or not vulnerabilities are exploited, we know the true labels $\{L\}_{t+1}^{t+T}$ of the vulnerabilities. With their true labels, we can now include these vulnerabilities $\{V\}_{t+1}^{t+T}$ in the training set $\{V, L\}_t^{t+T}$ and rebuild a new classifier C_{t+T}.

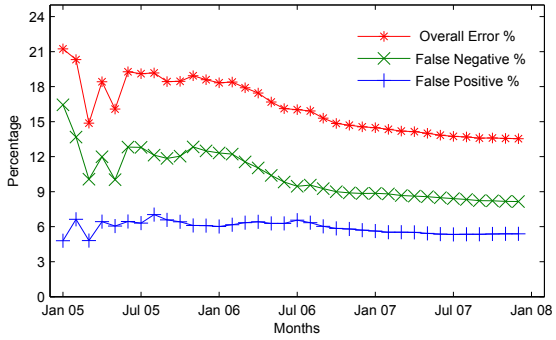
4. Calculate error — We also calculate the cumulative error of the classifier C_t. For each time interval of duration T, we count the number of predicted labels $\{L'\}$ that differ from their true labels $\{L\}$ in that interval, and sum all of the counts across all intervals. We then divide the sum by the total number of vulnerabilities seen up until that time. Calculating the cumulative error shows the stability of the classifier over time.

Figure 2 shows the results of this experiment. We train a classifier starting at January 2005 and initialize it with all prior vulnerabilities appearing before 2005. We then emulate the appearance of vulnerabilities and online reclassification and prediction through December 2007 (the end of our data set). We evaluate two update intervals T, once a week (Figure 2a) and once a month (Figure 2b). We show three curves for the total classification error as well as the false positive and negative rates over time.

These results show that, after initial fluctuations, the classifier stabilizes and improves its accuracy with more examples over time. At the end the classifier has an overall error rate of 14%, a false negative rate of 9% and a false positive rate of 5%. Further, classification accuracy is relatively insensitive to the update period: when the classifier stabilizes, the weekly and monthly results differ very little. These results demonstrate the viability of deploying a classifier in an online setting to predict whether vulnerabilities will be exploited.



(a) Weekly Training



(b) Monthly Training

Figure 2: Cumulative error, false negative, and false positive percentages for predicting whether vulnerabilities will be exploited in an online, deployed setting. We evaluate two time intervals for updating the classifier, every (a) week and (b) month.

5.5 Predicting Time to Exploit

Next we use SVMs to predict other metrics that help assess the severity of vulnerabilities. In practice, in addition to knowing whether a vulnerability will be exploited, it is also useful to know how soon it will be exploited. (Even if all vulnerabilities will eventually be exploited, it is valuable to know when.) With this knowledge, software vendors can prioritize the patches they release; system administrators can similarly prioritize the installation of these patches.

In general, there are three kinds of time-dependent exploits: positive-day exploits where an exploit is reported after the vulnerability is disclosed; 0-day exploits where an exploit is reported at the same time that the vulnerability is disclosed; and negative-day exploits where the exploit precedes the vulnerability disclosure date (e.g., an attacker exploits a vulnerability before the software vendor realizes the existence and nature of the vulnerability). Ideally, for each vulnerability, we would like know the probability distribution over days when it will be exploited. Such a distribution cannot

t (days)	$\|P\|$	$\|N\|$	Bias	SVM
2	1,404	2,632	65.21%	78.01%
7	1,960	2,076	51.44%	75.78%
14	2,330	1,706	57.73%	77.06%
30	2,733	1,303	67.71%	79.82%

Table 6: Predicting whether vulnerabilities will be exploited within t days.

be modeled by SVMs, which are designed for binary classification. However, we can use SVMs to make similarly relevant predictions.

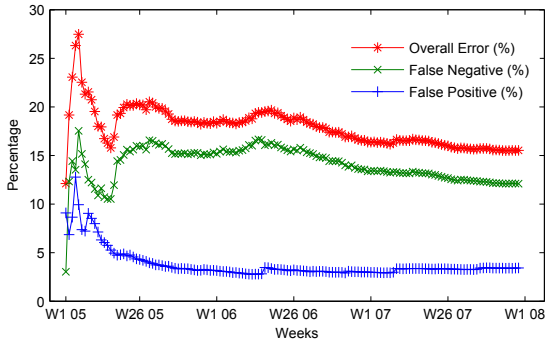
Next we use SVMs to predict whether or not a vulnerability will be exploited within some time t, where t is the difference between the exploit and disclosure dates of the vulnerability. To train such SVMs, we use the same examples as before, merely altering the labels to reflect whether a vulnerability has been exploited within some time frame (as opposed to whether it has been exploited at all). The set of "positive" examples P contains all vulnerabilities with positive-day exploits that are exploited within time t. The set of "negative" examples N contains vulnerabilities with positive-day exploits that are not exploited within time t. (Those that have 0-day or negative-day exploits need no prediction since their reports arrive with the vulnerability already exploited.) We then evaluate a range of time frames for t, from two days to one month.

For this experiment, we used an additional source of information with more accurate dates of vulnerability events. (Unfortunately, the OSVDB database has mixed-quality date information.) From his recent work developing a detailed empirical model of the vulnerability discovery, disclosure, and patch process [10], Stefan Frei generously shared the date information on vulnerabilities with CVE identifiers from his carefully collected data sets. We incorporated his data on discovery, exploit, and disclosure dates for the vulnerabilities contained in our data sets (Table 1).

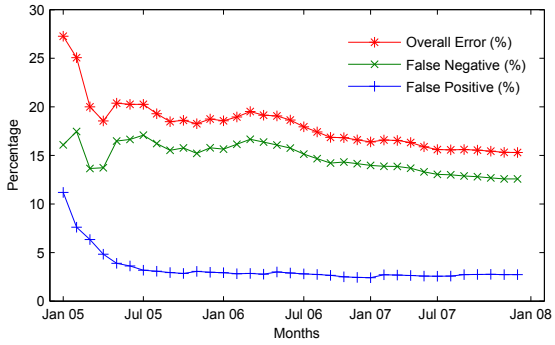
To evaluate the accuracy of predicting time-to-exploit for vulnerabilities, we perform both offline and online experiments similar to those in Sections 5.2 and 5.4. In the offline experiment we train and test classifiers on the entire data set, and in the online experiment we retrain the classifiers and make predictions on vulnerabilities over time.

Table 6 shows the results of the offline experiment. As in Section 5.2, we partition the examples into training and testing sets and report averaged results from cross-validation with ten different random partitions. For each experiment in the table, we show the predicted time frame t, the number of positive $|P|$ and negative $|N|$ examples, the accuracy $\max(|P|,|N|)/(|P|+|N|)$ of the default classifier that always predict the dominant label, and the accuracy from SVMs. The predictions from SVMs are 75–80% accurate across the different time frames; note that these results are significantly better than the raw bias induced from the imbalance of positive and negative training examples. Considering that we have not tuned the classifier, features, or thresholds to optimize the accuracy for this scenario, we believe that these results demonstrate the viability of predicting time-to-exploit from statistical analyses of vulnerability disclosure reports.

Figure 3 shows the results for the online version of the experiment. In the online version, we emulate a real-world deployment where we dynamically update the classifier and make predictions over time as new vulnerabilities appear. We show the results for predicting whether a vulnerability will be exploited within $t = 2$ days, the most severe positive-day case. (Other time frames, not shown, yielded similar results.) The classifier fluctuates initially, then stabilizes after training on a sufficient number of examples. The long-term trend shows a decrease in the false negative and



(a) Weekly Training



(b) Monthly Training

Figure 3: Cumulative error, false negative, and false positive percentages for predicting time to exploit in an online, deployed setting. We evaluate two time intervals for updating the classifier, every (a) week and (b) month.

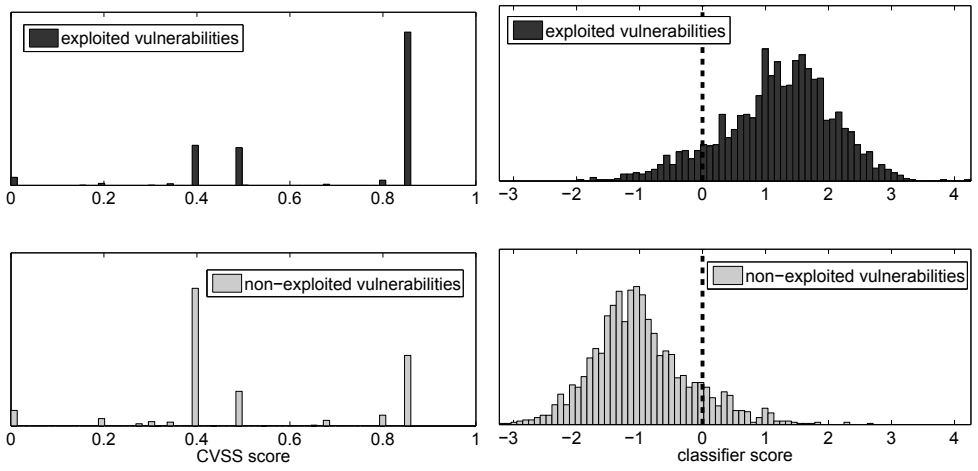
cumulative error rates while the false positive error rate remains flat. For a simple linear classifier, the overall results are extremely promising: at the end of training, the classifier has an overall cumulative error rate of 15%. Finally, in terms of errors, there are many more false negatives (13%) than false positives (2%).

5.6 Exploitability Metrics

Finally, we consider the issue of scoring metrics for vulnerabilities. Specifically, we compare two metrics for assessing how likely a reported vulnerability is likely to be exploited: one based on prior (expert) knowledge and handcrafted formulas, the other based on statistical methods and data mining.

As discussed in Section 2.2, the Common Vulnerabilities Scoring System (CVSS) defines a metric for scoring the "Exploitability" of a vulnerability; see eq. 2. We use this CVSS score as a representative formula-based metric. To be fair, the CVSS specification does not state how to interpret the "Exploitability" score; its intended purpose may not have been to represent the likelihood that a vulnerability is exploited. However, given its name and the factors that determine the score — e.g., difficulty and complexity of programmatically accessing the vulnerability in an exploit attempt — it seems reasonable to expect that the score correlates with exploit likelihood.

Our data-driven approach to vulnerability classification suggests an alternative scoring method. Recall that the decision rule in



(a) CVSS (b) Classifier

Figure 4: Histograms of exploitability scores computed on the vulnerabilities in our data set: (a) computed using the CVSS "Exploitability" formula (Eq. 2) with values normalized to 1; (b) computed using the classifier score ($\mathbf{w}\cdot\mathbf{x}+b$).

eq. (3) computes the signed distance to the maximum margin hyperplane separating positive and negative examples. The signed distance ($\mathbf{w}\cdot\mathbf{x}+b$) serves as a natural score for the exploitability of a vulnerability: the sign predicts whether it will be exploited, and for positively labeled examples, the magnitude indicates its severity.

We compare the effectiveness of these scoring methods by illustrating the distributions of their scores computed on the vulnerabilities in our data set. Visually, these distributions tell a compelling story.

Figure 4(a) shows histograms of CVSS "Exploitability" scores for exploited vulnerabilities (top) and vulnerabilities without exploits (bottom); we have normalized the scores to a maximum of 1. Note that CVSS automatically assigns a normalized score of 1 to all newly discovered vulnerabilities as a precautionary step. Vulnerabilities with that default score dominate the distribution, though, so we have removed them from the histogram to more clearly show the distribution of values computed by the CVSS formula. Figure 4(a) suggests that the CVSS exploitability scores on known vulnerabilities do not consistently reflect what happens in practice. Many vulnerabilities without exploits have high CVSS scores, and many vulnerabilities with exploits have low CVSS scores. As a result, no threshold CVSS score can differentiate well between the exploited and non-exploited vulnerabilities.

Figure 4(b) shows histograms of the classifier scores (i.e., the signed distances $\mathbf{w}\cdot\mathbf{x}+b$) for the same vulnerabilities. The vertical dashed line indicates the default threshold of zero used to predict whether a vulnerability should be labeled as "exploited" or not: values above the threshold are predicted as "exploited", and values below the threshold as "not exploited". As suggested by the results in Section 5.2, the histograms show that the classifier separates these distributions well: few exploited vulnerabilities have scores below the threshold (the false negatives), and few non-exploited vulnerabilities have scores above the threshold (the false positives). We note that preceding experiments included the CVSS score as a feature since it is available when a vulnerability is reported. However, we found that excluding the CVSS score as a feature did not noticeably change any of the results.

Overall, our results suggest that the security community should consider statistical models in addition, or as an alternative to current scoring practices. Such models have many compelling features. First, with little tuning, standard models such as SVMs can provide metrics that correlate well with exploit behavior. Second, the models can dynamically adapt over time to incorporate new features and data sets. Third, such models can be flexibly adapted to yield a variety of predictions—for example, whether a vulnerability will be exploited, or in what time frame it will be exploited. Fourth, the models provide real-valued scores that practitioners can use to prioritize vulnerabilities. Finally, these models can integrate the results from other scoring systems simply by incorporating the metrics defined by other systems as additional features used for classification.

6. CONCLUSION

Ranking vulnerabilities is a critical task for software companies. With thousands of vulnerabilities in hand and limited resources to fix them, it is important to prioritize any operational actions. Current methods, while easy to calculate, rely on static combinations of a small number of human-mediated qualitative variables that seem unlikely to capture the full complexity that drives vulnerability exploitation. In this paper we have described a complementary approach for vulnerability assessment using tools from data mining and machine learning. By considering a far broader range of features and relying on contemporary empirical data rather than "gut instinct" to determine their importance, we demonstrate that this approach can classify vulnerabilities significantly better than at least one currently (and widely) used system for severity scoring.

In general, we believe that machine learning is well-suited to many such security assessment tasks and offers considerable flexibility for consolidating disparate data sources so long as desirable security outcomes can be identified. For example, while this paper has focused specifically on exploitability, it would be straightforward for software vendors to use our approach in triaging *discovered* vulnerabilities to determine how to prioritize the development and deployment of patches. Finally, one limitation with existing vulnerability scoring approaches is they are generally "one size fits all"; they do not provide an easy mechanism for incorporating environment or context-specific information (aside from manually ad-

justing the ad hoc magic numbers in the formulas). In contrast, our data-driven approach provides a consistent way to integrate many local data sources, such as vulnerability scanners, IDS logs and incident ticketing systems, to specialize vulnerability assessment to a particular organization.

For many years, security assessment activities have been more art than science. While we concede that the "holy grail" security metric remains elusive, we see no reason to ignore the power of well-founded statistical methods that can improve the state of the practice.

Acknowledgments

We gratefully acknowledge the assistance of Stefan Frei, who generously shared his carefully collected data on vulnerability event dates [10].

7. REFERENCES

[1] W. A. Arbaugh, W. L. Fithen, and J. McHugh. Windows of vulnerability: A case study analysis. *Computer*, 33(12):52–59, 2000.

[2] A. Arora, A. Nandkumar, and R. Telang. Does information security attack frequency increase with vulnerability disclosure? an empirical analysis. *Information Systems Frontiers*, 8(5), 2006.

[3] A. Arora, R. Telang, and H. Xu. Optimal policy for software vulnerability disclosure. In *Workshop on Economics and Information Security (WEIS'04)*, 2004.

[4] S. M. Bellovin. On the Brittleness of Software and the Infeasibility of Security Metrics. *IEEE Security and Privacy*, 4(4), July 2006.

[5] Cisco. Risk Assessment: Risk Triage for Security Vulnerability Announcements. Cisco Whitepaper, Accessed September, 2009. http://www.cisco.com/web/about/security/intelligence/vulnerability-risk%-triage.html.

[6] CVE Editorial Board. Common Vulnerabilities and Exposures: The Standard for Information Security Vulnerability Names. http://cve.mitre.org/.

[7] C. Dougherty. Vulnerability metric, Updated on July 24, 2008. https://www.securecoding.cert.org/confluence/display/seccode/Vulnerabil%ity+Metric.

[8] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR – A Library for Large Linear Classification. http://www.csie.ntu.edu.tw/~cjlin/liblinear/.

[9] Forum of Incident Response and Security Teams (FIRST). Common Vulnerabilities Scoring System (CVSS). http://www.first.org/cvss/.

[10] S. Frei, D. Schatzmann, B. Plattner, and B. Trammel. Modeling the Security Ecosystem — The Dynamics of (In)Security. In *Proc. of the Workshop on the Economics of Information Security (WEIS)*, June 2009.

[11] IBM. IBM Internet Security Systems X-Force 2008 Trend and Risk Report. White paper, Jan. 2009. http://www-935.ibm.com/services/us/iss/xforce/trendreports/xforce-2008-%annual-report.pdf.

[12] D. Lewis. Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval. In *Proceedings of ECML-98, the 10th European Conference on Machine Learning*, pages 4–15, 1998.

[13] P. Mell, K. Scarfone, and S. Romanosky. A complete guide to the common vulnerability scoring system version 2.0, June, 2007. http://www.first.org/cvss/cvss-guide.html.

[14] Microsoft TechNet Security Team. Microsoft Security Bulletin. http://www.microsoft.com/technet/security/current.aspx.

[15] D. Moore, C. Shannon, and k. claffy. Code-red: a case study on the spread and victims of an internet worm. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 273–284, 2002.

[16] D. Nizovtsev and M. Thursby. Economic analysis of incentives to disclose software vulnerabilities. In *Proc. of the Workshop on the Economics of Information Security*, 2005.

[17] OSVDB. The Open Source Vulnerability Database. http://osvdb.org/.

[18] A. Ozment. The likelihood of vulnerability rediscovery and the social utility of vulnerability hunting. In *Proc. of the Workshop on the Economics of Information Security*, 2005.

[19] E. Rescorla. Security holes... who cares? In *Proc. of the 12th conference on USENIX Security Symposium*, 2003.

[20] Secunia Corporation. Secunia Advisories. http://secunia.com.

[21] Symantec Corporation. Security Focus. http://www.securityfocus.com.

[22] V. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, New York, NY, 1998.