Technical Report AI-06-335, Artificial Intelligence Lab, The University of Texas at Austin, August 2006.

Fast and Effective Worm Fingerprinting via Machine Learning

Stewart Yang, Jianping Song, Harish Rajamani, Taewon Cho, Yin Zhang and Raymond Mooney
Department of Computer Sciences, University of Texas at Austin, Austin, TX 78705, USA
{windtown, sjp, harishr, khatz, yzhang, mooney}@cs.utexas.edu

Abstract

As Internet worms become ever faster and more sophisticated, it is important to be able to extract worm signatures in an accurate and timely manner. In this paper, we apply machine learning to automatically fingerprint polymorphic worms, which are able to change their appearance across every instance. Using real Internet traces and synthetic polymorphic worms, we evaluated the performance of several advanced machine learning algorithms, including naive Bayes, decision-tree induction, rule learning, and support vector machines. The results are very promising. Compared with Polygraph, the state of the art in polymorphic worm fingerprinting, several machine learning algorithms are able to generate more accurate signatures, tolerate more noise in the training data, and require much shorter training time. These results open the possibility of applying machine learning to build a fast and accurate online worm fingerprinting system.

1. Introduction

One of the major goals of autonomic computing is to make computers self-protecting. Computer worms [22] are programs that exploit security flaws in existing computer systems. Like viruses, worms employ compromised hosts to carry out assigned tasks; unlike viruses, worms aggressively leverage resources of victims to propagate themselves across the network and compromise more vulnerable hosts. Although not all worms are malicious, the aggressive propagation of worms can increase Internet traffic tremendously in a short period and thus slow down normal traffic or even disable large areas of the network. In fact, the past decade has seen an increasing number of worm outbreaks, among which the most deadly ones caused billions of dollars in loss. For example, the Code Red worm alone led to 2.6 billion dollars losses in revenue and productivity during the first week of its outbreak [13].

Given the severity of the problem, researchers have developed various systems and approaches to discover and quarantine worms. Most conspicuously, existing intrusion detection systems such as [16, 17] have been employed to quarantine worms whose signatures are known. A typical intrusion detection system will monitor all the incoming and outgoing traffic, while removing traffic flows that match predefined rules (signatures). These systems can scale up easily by introducing parallelism. However, the problem is not yet solved, because worm fingerprinting currently requires security experts to manually analyze captured worm instances and thus can be very slow [14]. Meanwhile, recent studies have shown that new worms such as the SQL SLammer can compromise all vulnerable hosts in the network in as short as 10 minutes [12]. As a result, in order to effectively stop worm outbreaks, new automated worm fingerprinting techniques need to be developed.

Worms released in the past few years have become even more powerful and deadly by using polymorphic techniques to prevent themselves from being detected [9]. To face this challenge as well as the time challenge mentioned above, automatic worm signature generation techniques have been developed to replace the current manual fingerprinting approach. The aim of these techniques is to enable a computer system to discover and generate signatures for new worms without human intervention. Advances achieved in this direction will contribute sinificantly to the self-protecting aspect of autonomic computing.

For example, Autograph [8], is an automatic worm fingerprinting system. Autograph sits on the border between networks and monitors the through traffic. It uses a heuristic approach to collect suspicious flows and then constructs signatures from those flows using a greedy induction algorithm. The authors have shown that this system correctly generates signatures for two famous worms, Nimda and CodeRed. Polygraph [15] is another system that builds on the Autograph framework, but focuses on fingerprinting polymorphic worms. The authors proposed three approaches to generating signatures: sequential, conjunctive and Bayesian. Experiments showed that all three approaches worked well for fingerprinting two polymorphized worms, though fingerprinting speed was still a concern.

In this paper, we examine the applicability of vari-

ous machine-learning algorithms to automatic worm fingerprinting. Numerous alorithms have been developed for learning classifiers from labeled training data. Because the task of worm fingerprinting is essentially creating patterns for worm flows that can be used to classify future flows as malicious or innocuous, we argue that many well established classification algorithms can be applied to solve the problem. These algorithms have been designed for classification accuracy as well as other properties, such as speed of training and tolerance to noise in the training data.

We focused our study on fingeprinting polymorphic worms and compared five existing learning algorithms with Polygraph, which to our knowledge is the only existing system designed specifically for this task. Experiments were conducted on recorded real traffic data. For each algorithm, we evaluated its false positive rate (misclassifying innocuous as malicious), false negative rate (misclassifying malicious as innocuous), and speed of training. Experimental results show that an existing decision-tree learning algorithm and a rule learner both run faster than Polygraph and produce fewer errors.

The remainder of this paper is oranized as follows. In Section 2, we discuss previous work in more detail. In Section 3, we provide further motivation for our work and introduce the four learning algorithms we have evaluated. Section 4 presents the design and results of our experiments. Section 5 presents our conclusions and directions for future work.

2. Background on Existing Work

Worm detection, and more generally intrusion detection, has been studied fairly extensively in the past few years. Algorithms in the area can be roughly organized into four categories. The first family of algorithms produces rules that match malicious or suspicious behaviors by analyzing traffic statistics[16, 17], and thus is called *traffic-based algorithms*. For example, a rule can be "all flows from host A's port B are malicious and shall be filtered". The second approach, usually called *honeypots*[21], involves setting up virtual hosts on unused IP addresses in the network which silently record all the incoming flows. Because those IPs are unused, incoming traffic flows can be simply regarded as suspicious. By manually or automatically analyze those flows, new worms can be discovered. The third approach, usually referred to as *behavior-based algorithms*[7], works by monitoring vulnerable hosts' behaviors. Alarms are raised when the behaviors become abnormal or when the hosts fail to pass integrity checks. The last method, *content-based algorithms* [8, 15], works by analyzing the content of suspicious flows from and to the vulnerable hosts. First, these flows are roughly classified as unsuspicious or suspicious, and then signatures are constructed to identify the suspicious flows.

While there is no previous study comparing all the four families of algorithms, we argue that the *content-based algorithms* are the best choice because of their appropriate balance of efficiency and effectiveness. Among the four family of algorithms, *traffic-based*, *behavior-based*, and *content-based* algorithms can all run automatically. In particular, *content-based* algorithms can be seen as standing between *traffic-based* and *behavior-based* algorithms. Unlike *traffic-based* algorithms, signatures produced by *content-based* algorithms are usually specific enough to produce few false positive alarms in production. At the same time, *content-based* algorithms avoid the high cost of augmenting server software for the purpose of monitoring, as well as the overhead caused by these augmentations at runtime, which are all required by the *behavior-based* algorithms and can easily be formidable in practice. Finally, the limitation of the *honey-pot* approach is clear – the amount of traffic entering honeypots is limited due to the increasingly populated IP address space, thus it may take too long to gather enough worm samples for effective fingerprinting.

Among *content-based* intrusion detection systems, Autograph [8] was the first to explicitly target worms. When deployed on the border of a network, the system uses a predetermined heuristic to identify suspicious hosts and mark all traffic from these hosts as suspicious, leaving other flows as unsuspicious. The suspicious and unsuspicious flows are then fed as training data to the fingerprinting algorithm to generate accurate worm signatures. Although at first glance this setting may look odd since both the heuristic and the fingerprinting algorithm have the same goal, i.e. distinguishing worm flows from innocuous flows; in reality they serve very different purposes. The heuristic is a rough preprocessing step and thus is not required to achieve high accuracy, namely, the suspicious flows collected by the heuristic may contain many inocuous flows or miss worm flows. Meanwhile, the fingerprinting algorithm uses its results as noisy training data to generate more accurate signatures to be used in practice. As a result, the fingerprinting algorithm needs to achieve high accuracy as well as being noise-tolerant.

With the labeled training data, the next step is to construct features from the payloads of these flows. Autograph uses Robin's fingerprint algorithm [19] to do this. Briefly, Robin's fingerprint algorithm separates a payload string into many non-overlapping substrings according to a pre-selected terminal symbol; these substrings are then used as features to represent the original flow. Eventually, suspicious flows represented by substring features are presented to a greedy signature generator. The generator recursively finds the feature present in largest number of remaining flows, makes a rule that this feature implies a worm, and removes all flows that match this rule. This procedure con-

tinues until a predefined percentage of suspicious flows is covered, or there are no more features to select. The resulting set of rules is then deployed in an intrusion detection system to filter out worm flows.

Experimental evaluation of the Autograph algorithm was quite promising. On two traffic traces recorded at Intel and ICSI, which contain instances of the Nimda worm and the CodeRed II worm, Autograph produced signatures that successfully identified all worm flows. Moreover, the algorithm gave few false positive alarms, as all the 17 signatures generated were indeed true worm signatures. However, if more innocuous flows made their way into the suspicious pool, it is unknown whether the algorithm will be resilient and not produce false alarms. Although the authors proposed a blacklist method to handle this problem, which is essentially manual removal of signatures that produce false alarms, the cost and delay caused by such human intervention would be problematic.

Along with improved worm fingerprinting techniques, researchers have also proposed methods that could be adopted by worm authors to enhance their future worms. In particular, various techniques can be employed to "polymorphize" a worm and thus allow it to evade current detection techniques [9]. For example, a polymorphic engine (e.g. [20]) can encrypt the original payload and then link it to a decryptor, so when this polymorphic payload reaches its destination, the original payload can be extracted and executed as normal. However, these polymorphized worm flows, when presented to a worm detection system like Autograph, may prevent it from producing any signatures/alarms. Besides encryption, other techniques such as register swapping and no-op insertion serve the purpose as well.

Fortunately, it has been noted by Newsome et al. [15] that current polymorphic engines will always leave a few invariant chunks scattered around the encrypted body. Based on this observation, they proposed three algorithms which focus on detecting and generating signatures for polymorphic worms. Named after Autograph, these algorithms were collectively called Polygraph. Polygraph has roughly the same system architecture as Autograph; however, Polygraph employs a common substring finder instead of Rabin's fingerprint algorithm to construct features. The authors argued that because a common substring finder favors popular flows, potential signatures of a new worm will be extracted quickly in the early moment of its outbreak.

Using the extracted features, Polygraph employs three different algorithms for signature generation. The first sequential-signature algorithm defines signatures as sequences of features. It starts from the most specific feature sequences (individual flows in the suspicious pool) and iteratively combines the two most similar set of feature sequences into a new signature by retaining the longest common subsequence of the two and substituting the rest with wildcard characters. This process stops when the new signature is so general that it matches more than a given number of innocuous flows in the validation set, and then all signatures generated to this point are output as worm signatures. Similar to this algorithm, a second conjunctive-signature algorithm employs the same steps but generates signatures as conjunctions of features, removing the locational information. The third algorithm applies Bayesian technique to assign each feature a weight to indicate how important it is in deciding the nature of the flow.

Similar to Autograph, experiments on Polygraph were carried out on recorded traces. Meanwhile, instead of detecting real worms, the algorithms were configured to detect simulated worm flows, which were generated by filling random characters into real worm signatures. Experimental results showed that Polygraph was very resilient to polymorphism in most cases. All three algorithms worked perfectly when the noise level in the suspicious pool was low. As the noise level increased, the Bayesian algorithm began to have considerable false-positives and false-negatives, but the sequential and conjunctional algorithms still maintained zero false negative rates and low false positive rates. However, it was also noted that both the sequential and the conjunctive algorithm took "less than ten minutes" to fingerprint a suspicious pool of just 25 flows. Actually, these two algorithms are of $O(n^2m)$ complexity, in which n and m are the numbers of suspicious and unsuspicious flows respectively. Under the current condition that a worm can breach all vulnerable hosts in 10 minutes, this speed is insufficient to effectively quarantine or stop a new worm early in its outbreak. As a result, new algorithms are needed which run faster and scale to larger number of suspicious flows, which is a main focus of this paper.

Finally, it is worth mentioning a recent study [10] on applying machine learning to fingerprint malicious executables, which include viruses, worms and Trojan horses. The authors gathered 1,651 malicious executables and 1,971 benign executables on the Windows platform. After turning core dumps of these executables into an n-gram feature representation, the authors applied an array of learning methods to construct classifiers that predict whether an executable is malicious or benign. Among the algorithms tested, boosted decision trees performed the best in terms of accuracy. Nonetheless, all algorithms generated satisfactory Receiver Operating Characteristic (ROC) curves. While this is an interesting study, it is significantly different from ours in three aspects. First, the target in our study is Internet traffic flows instead of executable binaries and we focus solely on worms; second, while speed was not an issue in detecting malicious executables offline, it is of great importance here because it is the key to early quarantine of a worm in its outbreak. Finally, rather than building a classi-

fier on training data free of class noise, our problem involves generating signatures on potentially very noisy data.

3. Worm fingerprinting via machine learning

The task of worm fingerprinting can be abstracted as follows: given a few labeled training examples, construct a classifier to separate a specific type of flow (worms) from all other flows (innocuous flows) based on their content. Additionally, an ideal classifier should be fast to train and resilient to noise (e.g. misclassifications) in the training examples. Under the current scheme, knowledge embedded in the learned classifier can be extracted as explicit signatures and passed to an existing intrusion detection system, which matches incoming flows against these signatures and removes the matching ones. Alternatively, these classifiers could be directly employed by an intrusion detection system to assign a malicious or innocuous label to incoming flows.

A large number of classification algorithms exists in the machine learning literature and are actively employed in various applications. In general, these algorithms optimize for calssification accuracy, the percentage of instances that are correctly classified. However, learning algorithms have been designed to simultaneously optimize for other criteria, such as training time and noise resistance. In particular, in terms of time complexity, most of the methods tested here are linear in the size of the training data, compared to the higher complexity of Polygraph. Most have also been explicitly designed to handle noisy training data.

As the main contribution of this paper, we conducted extensive experimental studies to verify our conjecture that other standard machine learning methods would outperform those used in Polygraph. The algorithms we tested were used "right out of the box" from the Weka data-mining package [23], except for sequential-signature Polygraph, which we implemented following [15]. We chose the sequential-signature version of Polygraph as the baseline since it was shown to be significantly more accurate than the other versions when there was any noise in the suspicious pool [15]. In the remainder of this section, we briefly introduce the five learning algorithms we evaluated.

3.1. Naive Bayes learners

Bayesian learning algorithms are founded on Bayes theorem, which, in the context of classification, states that the posterior probability of a class is proportional to its prior probability as well as the conditional likelihood of the features given this class. If no independence assumptions are made, a Bayesian algorithm must estimate conditional probabilities for an exponential number of feature combinations. "Naive Bayes" simplifies this process by making the assumption that features are conditionally independent given the class and requires estimating only a linear number of parameters. The prior probability of each class and the probability of each feature given each class is easily estimated from the training data and used to determine the posterior probability of each class given a set of features. Empirically, Naive Bayes has been shown to produce good classification accuracy across a variety of problem domains [4]. The simplicity of its implementation and its fast (linear) training time has made this algorithm a popular choice in practice.

In this study, we evaluated two versions of Naive Bayes, the standard version that comes with Weka (NB) and a Multinomial Naive Bayes (MNNB). Initially proposed for text classification [11], MNNB assumes a multinomial distribution for real-valued features instead of a Gaussian distribution. It has been observed to outperform standard Naive Bayes for text classification. Also, it is worth noting that the Bayesian-signature Polygraph differs significantly from the above algorithms by adopting an non-standard way of calculating feature likelihoods.

3.2. Support vector machines

Support vector machines (SVMs) [3] were introduced in the mid-90s and experiments have subsequently shown them to be the most accurate current classifiers in a variety of applications. The algorithm typically projects the original feature vectors into a higher dimensional space and then tries to find a hyper-plane in that space that best separates the two classes of instances. Facilitated by a *kernel*, which computes the dot product of two feature vectors in the high dimensional space directly from the original feature vectors, the computation required by the projection and optimization is greatly simplified and made computationally tractable. Specifically, we employ the SMO algorithm for SVMs implemented in Weka. Over the years, researchers have developed various kernels that map the original feature vector into different high-dimensional spaces. Among these functions, the most frequently used ones are the linear kernel, the polynomial kernel, and the RBF kernel. In this paper, we present results with an RBF kernel since it performed the best.

3.3. Decision tree learner

Decision tree learners [18] are a well-established family of learning algorithms. Initially proposed in the 70s, these algorithms have been continuously developed to include more features and yield better performance. Classifiers are represented as trees whose internal nodes are tests on individual features and whose leaves are classification decisions. Typically, a greedy heuristic search method is used to find a small decision tree that correctly classifies the training data. In order to handle noisy data, they are

typically augmented with a "pruning" procedure that prevents overfitting the training data. In this study we evaluated J48, the Weka version of the commonly used C4.5 algorithm [18]. Various studies in the past have shown that it is an efficient algorithm that learns accurate classifiers in many domains.

3.4. Rule learner

Rule learners [2] also originated in the 70's and induce a set of if-then rules with conjunctive premises. A typical rule learner also uses greedy search to learn a small set of rules consistent with the training data. The process proceeds in iterations, each time a rule is constructed to match as many instances of the minority class as possible and those instances are removed from the training set. The learner keeps producing rules until all remaining training instances belong to one class. The construction of each rule is done in a similar manner – premises are continually added to the rule until the rule only matches instances of a single class. Rule learners also include various pruning methods to avoid overfitting the training data and make the learner robust to noisy data. RIPPER [2] is a fast effective rule learning algorithm that is particularly suitable for noisy training data. In our study, we used the Weka implementation of RIPPER, which is called JRip.

Among the five algorithms introduced in this section, the rule learner is the most similar to Polygraph's sequential signature algorithm. Both learn a pattern in the form of a disjunction of conjunctions of features. However, the Polygraph algorithm has a time complexity of $O(n^2m)$, whereas RIPPER leverages the extensive body of research on rule learning and has a time complexity of just $O(m+n)$. Moreover, while the Polygraph has some ad-hoc methods for tolerating noise, RIPPER includes a general, carefully developed and evaluated method for preventing over-fitting. The advantages of these advanced machine-learning techniques are illustrated in the results.

4. Experimental results

4.1. Experimental Design

Our experimental comparisons were conducted on a combination of network traffic traces and self-generated polymorphic worm instances. The two network traces were collected from a 100Mbps fiber link at ICSI. In particular, one trace was recorded during the one week span from May 22nd to May 29th, 2004 and will be called the week trace; the other trace was recorded for one day on Jan 26th, 2004 and will be called the day trace. These two traces were previously used in experiments on Autograph. As preprocessing, we reassembled packets in the two traces into flows and filtered out flows that were labeled as worms by the Bro intrusion detection system [16], thus the resulting pool of flows only contained innocuous flows. Following the studies on Polygraph, we generated polymorphic worm flows for the Apache-knacker worm and the Atphttpd worm. Headers of these worm flows were sampled uniformly from the pool of previously constructed innocuous flows, and bodies of these flows were constructed from known signatures of these two worms by filling random characters into the wildcard slots of these signatures.

As the next step, we transformed the string-based flows into feature-vector representations. We employed two feature construction techniques: a common substring finder like that used in [15] (COM) and an n-gram finder like that used in [10] (n-GRAM). COM looks for substrings within a predetermined length limit that appear in more than a given percentage of flows. More specifically, the algorithm starts with the longest common substrings and turn them into features. It then moves on to second-to-longest common substrings, but this time prevalence counting excludes appearances of those substrings that form part of any existing features. This process repeats until the minimum-length limit is reached. As in Polygraph, we implemented COM based on the suffix tree data structure [6], with a time complexity of $O(ml)$, where m is the number of flows and l is the average length of a flow. An n-gram of a given payload is defined as a substring of n characters that occurs anywhere in the payload. Like the approach to feature construction in [10], n-GRAM finds all n-grams in the payloads and retains the 500 n-grams with the highest information gain with respect to discriminating between suspicous and unsuspicuous flows. In order to find the best parameters for the two methods, we conducted development experiments on the day trace. For COM, we varied the minimum-length limit from 2 to 10 incrementing by 2 and the prevalence limit from 2% to 10% incrementing by 2%, and chose for each learning algorithm the pair of parameters that yielded the most accurate signatures. For n-GRAM, we varied the minimum-length limit from 2 to 10 incrementing by 2 and also chose the best value for each learning algorithm.

The experiments were carried out on desktop machines with 3.0GHz Intel Pentium IV processors and running Linux kernel 2.6.13. We compared all six algorithms based on the following criteria. To measure the accuracy of generated signatures, we recorded the cross-validated false positive rate (the percentage of innocuous flows incorrectly classified as worms) as well as the false negative rate (the percentage of worm flows misclassified as innocuous). To evaluate the resilience of these algorithms to unavoidable class noise in the suspicious pool, we computed noise curves by varying the ratio of innocuous flows in the suspicious pool from 10% to 90% incrementing by 10% each time and recording the error rates at each point. Finally, to evaluate

the detection speed, we recorded the time required by each algorithm to process the training sets. It is worth noting that except for the timing experiment in section 4.3.2, we did not include feature construction time. To ensure the reliability of the results, for each setting we report the results averaged over ten runs.

4.2. Accuracy of Generated Signatures

4.2.1 With a worm-free unsuspicious pool

Following the experimental design used to test Polygraph, we first evaluated the case when there are no worm flows in the unsuspicious pool. As suggested in [15], at any given time during production use, the unsuspicious pool used by a signature learner will be created from flows that were recorded a few days earlier. This setting will help ensure that in day zero of a new worm outbreak, the new worm will not appear in the unsuspicious pool. Although the suspicious pool may still contain innocuous flows, the following results show that most algorithms are able to fingerprint the two worms in this scenario.

Our experiments were carried out on the week trace plus one thousand simulated polymorphic worm flows. We tried suspicious pools containing 50, 100, 200, and 400 flows and recorded the results for each. For each suspicious pool size, we generated a noise curve for different levels of class noise as described below. This noise, namely innocuous flows mislabelled as suspicious, was drawn uniformly from the week trace flows. The unsuspicious pool size was fixed to contain 45,000 flows that were also drawn uniformly from the remaining week-trace flows. All the remaining week-trace flows and simulated worm flows were used to test the accuracy of the generated signatures. Although several experiments did not run to completion because of extremely long running time, in general we obtained consistent results for most cases. Because the pattern of results was quite similar across feature construction methods and different suspicious pool sizes, due to space constraints, we only present the results with COM features and 50 suspicious flows. Figure 1 shows the false positive and false negative rates for this case. In both graphs, the x-axis is the level of noise in the suspicious pool and the y-axis is the number of flows that are misclassified. Some curves overlap on the graph, for example, the false positive rates for Polygraph, JRip, J48 and SVM are all consistently zero.

From the graphs it is clear that the two Naive Bayes algorithms had the highest false positive rate as well as the lowest false negative rate. On the test data, they mislabeled over half of the innocuous flows as worms but classified almost all worm flows correctly. Naive Bayes' blanket assumption of conditional independence is known to frequently lead to incorrect probability estimates and classifications. The abundance of false postives could be reduced by requiring a

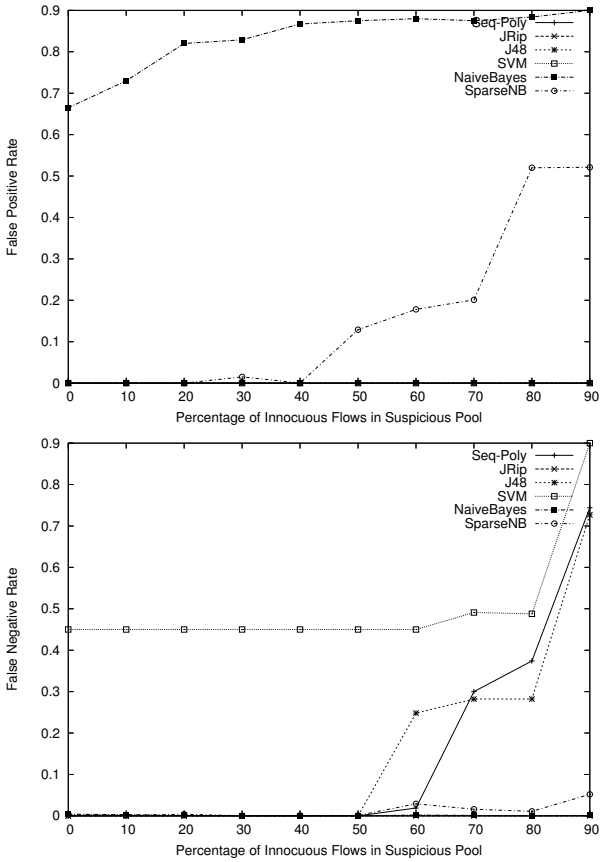

Figure 1. False positive and false negative rates under varying percentage of noise in the suspicious pool (50 flows). Unsuspicious pool (45,000 flows) contains no worm flows.

threshold above 0.5 on the posterior probability in order to assign a flow to the worm class; however, this would require determining an appropriate threshold and would risk resulting in false negatives. As a result, in the following studies, we exclude the Naive Bayesian algorithms from consideration.

Among the remaining algorithms, SVMs had a high false negative rate and failed to recognize half of the worm flows, though its false positive rate was acceptable. All of the other algorithms had low false positive rates and low false negative rates continuously along the noise curve. The only exception is that Polygraph started to generate false positives when the level of noise in the suspicious pool increased, which agrees with the results presented in [15]. JRip performed the best as it achieved zero false positive rates consistently; in fact, for most runs, JRip just produced a single rule "0xFF0xBF implies worm". It turns out that this two-byte substring is the entry address of the security flaw in the server system, which is required for any worm to break into

it. Moreover, this address is not seen in any of the innocuous flows. Even more impressive is the fact that JRip successfully generated this signature when there were only five true worm flows in a suspicious pool of size 50, which suggests it would be able to detect a new worm early in its outbreak. Since there seem to be small "smoking gun" signatures for such worms, it is not surprising that symbolic rule learning algorithms like JRip are more accurate than more numerical and probabilistic methods since their bias for finding simple symbolic descriptions of categories seems to be a good match for this problem.

Finally, it is worth mentioning that we also experimented with boosted and bagged version of the above algorithms. Boosting [5] and bagging [1] are two important ensemble methods in machine learning. By creating a diverse committee of base classifiers and combining their decisions, they usually improve predictive accuracy. However, our experiments found that these methods actually decreased accuracy on this problem. For boosting, we believe the reason is that it overfits the noisy training data; for bagging, it is probably because the number of worms in the training set is reduced substantially by its boostrap sampling.

4.2.2 When the unsuspicious pool contains worms

One major assumption made in both Polygraph and our previous experiments is that the unsuspicious pool is free of worm flows. The authors of Polygraph argued that this can be achieved by using flows from a few days ealier to form the unsuspicious pool. However, they also admitted that a worm author could break this premise by stealthily blending possible signatures of a new worm into innocuous traffic and wait a few days to launch the worm attack. We conjecture that in this case Polygraph will be less effective, because genuine worm signatures will seemingly yield false positives on the validation set. As a result, either these signatures are discarded, which leaves the new worm undetected, or the threshold for accepting signatures is lifted, which allows erroneous signatures to be created for noise in the suspicious pool. To verify our conjecture, we experimented with unsuspicious pools that contained worm flows. More specifically, we kept all settings the same as the previous experiments, except we blended twenty simulated worm flows into the unsuspicious pool. Because results are similar for different numbers of suspicious flows, those using a suspicious pool of size 200 are shown this time, and we only present results for algorithms that did well in the previous experiments.

As shown in Figure 2, Polygraph had a consistently high false negative rate. Even when the suspicious pool consisted of only worm flows, Polygraph mislabeled 58.5% of worm flows as innocuous. In contrast, JRip has a zero false negative rate when the suspicious pool is free of innocuous flows. It continued to generate zero false negative rates when the number of innocuous flows increased, and only began to mislabel worm flows as innocuous when the number of worm flows in the suspicious pool dropped below that of the unsuspicious pool. Together these results show that JRip is much more resilient to noise in the innocuous pool than Polygraph is. The performance of J48 stands between those of the above two algorithms, with an exception that J48 outperformed JRip when there were 20 worm flows.

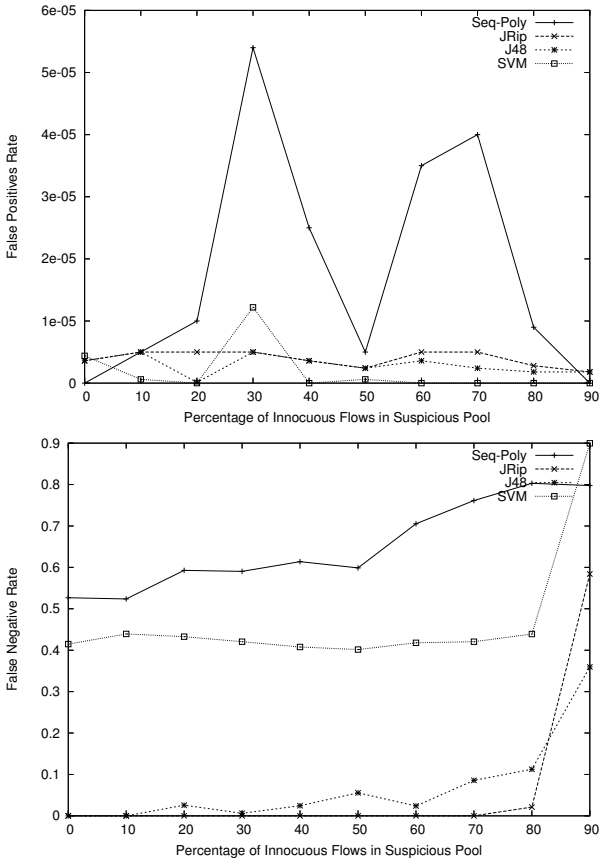

Figure 2. False positive and false negative rates under varying percentage of noise in the suspicious pool (200 flows). Unsuspicious pool (45,000 flows) contains 20 worm flows.

As stated earlier, worm authors can disable worm detection algorithms by poisoning the unsuspicious pool before launching the attack. On the other hand, the use of recorded innocuous flows to form the unsuspicious pool may cause problems even without adversarial attack. More specifically, as new web applications are being constantly introduced, the traffic flows produced by these applications may be quite different from those of existing applications. Meanwhile, because most methods for determining suspicious flows look for abnormalities, novel flows created by

the new applications are more likely to be labeled as suspicious. As a result, worm fingerprinting algorithms will generate erroneous worm signatures for the new legitimate flows right after their release, because these flows only exist in the suspicious pool and not in the unsuspicious pool. Even worse, if the erroneous signatures were automatically utilized by the intrusion detection system, legitimate traffic will be filtered out, which is extremely undesirable. Therefore, in order to prevent such a disaster, human experts must be deployed to manually look over signatures generated before actually deploying them, which will greatly slow down the process and thus reduce the possibility of quarantining a worm early in its outbreak.

We verified this conjecture by a simple experiment on the day trace. We created one hundred new and distinct flows by substituting the URL part of sampled innocuous flows with a unique string. In this way the flows remained legitimate but were different from other innocuous flows. Then we blended these new flows into the suspicious pool but not the unsuspicious pool, and reran the experiment. It turned out that all fingerprinting algorithms examined recognized these new and legitimate flows as worms. This result suggests to us that we may need to use same-day flows to form the unsuspicious pool. However, therefore it is impossible for us to guarantee that the unsuspicious pool consists solely of innocuous flows, instead, the best assumption we can make is that there are more worms in the suspicious pool than in the unsuspicious pool given a reasonable method for identifying suspicious flows. Consequently, resilience to noise in both the suspicious and unsuspicious pool is crucial for a worm fingerprinting algorithm to work well. As we have shown, learning algorithms like JRip clearly win in this regard.

The above results lead us to believe that standard machine learning algorithms, in particular, JRip and J48, are superior to Polygraph in terms of accuracy of signatures and resilience to noise.

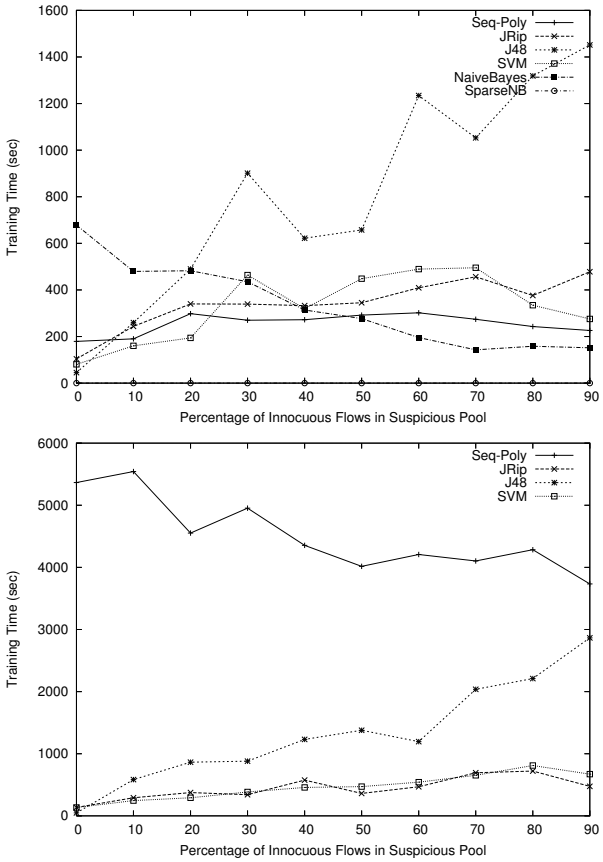

Figure 3. Training time under varying percentage of noise in suspicious pool. The above graph depicts suspicious pool of size 50 and pure unsuspicious pool. The bottom graph depicts suspicious pool of size 200 and unsuspicious pool with 20 worm flows

4.3. Training Time

4.3.1 Training time for the accuracy experiments

As previously stated, the time required to train a worm detector is a crucial factor in effectively quarantining new worms. Figure 3 presents depict training time for the two experiments presented in the previous subsection. It is worth noting that the training time for a pure unsuspicious pool does not differ much from that of an impure unsuspicious pool. Again, we focus on the algorithms that generated more accurate signatures, namely, JRip, J48, and Polygraph.

The time complexity for Polygraph is $O(n^2m)$, in which n is the number of suspicious flows and m is the number of unsuspicious flows. This can be clearly observed in the results, as when the size of suspicious pool grows to four times as large, the training time increases roughly sixteen times. For example, when there are 200 flows in the suspicious pool, Polygraph takes more than an hour to train. We believe this is one of the major limitations of the Polygraph algorithm, because in the outbreak of a new worm, the suspicious pool can easily grow up to hundreds or even thousands of flows, and consequently the time required by Polygraph to train on this suspicious pool will be too long to effectively quarantine the worm. On the contrary, the time complexity for JRip is $O(m + n)$, which is also observed in the graphs, as when the size of suspicious pool increase from 50 to 200, the training time stays roughly the same because the number of unsuspicious flows (45,000) dominates the total number of flows. In practice, this property is crucial to restricting the training time in a worm outbreak to a tractable level so that the resulting rules can be learned fast enough to quarantine the worm. Meanwhile, the train-

ing time of J48 lies between that of JRip and Polygraph. Though J48 can be seen as following roughly the same philosophy as JRip, the fact that its pruning is done only after the tree is fully grown makes it slower. Finally, it is worth noting that testing time, i.e. the time required to make predictions on testing instances, remains negligible for all these algorithms.

4.3.2 End-to-end training time for production use

Although JRip takes between 3 to 10 minutes to train regardless of the size of suspicious pool, it remains dubious whether this is still fast enough to effectively quarantine a worm outbreak. As a result, we conducted additional experiments to explore how fast the Ripper algorithm could train on the minimum number of examples necessary to identify a worm. More specifically, we used the original more efficient C implementation of the RIPPER algorithm instead of the JRip Java version, coded the n-gram feature extractor efficiently in C++, and streamlined the entire process from reading reassembled flow payloads to outputting the generated signatures. For feature construction, n-GRAM was chosen over COM because it leads to comparable signature quality with significantly less computation.

We then measured the end-to-end time required to fingerprint a new worm with this "production level" implementation of our approach. Moreover, as the fingerprinting algorithm may not need as many as 45,000 unsuspicious flows to contrast against the suspicious flows, we used the week trace but gradually lowered the number of unsuspicious flows to the minimum number needed to successfully fingerprint the worms, creating a learning curve. There were 200 suspicious flows, of which 50 were worm flows; meanwhile, in the varying number of unsuspicious flows there are consistently 20 worm flows. In our opinion, this configuration is a reasonable model of a real-world situation early in a worm outbreak.

The results are shown in figure 4. In addition to end-to-end training time, false positive and false negative rates are also plotted to present a more comprehensive view. Therefore, the x-axis represents the number of unsuspicious flows, while the left y-axis and right y-axis depict false positive/negative rate and end-to-end training time respectively. Finally, results of experiments with more than 2,000 unsuspicious flows are omitted because they follow the same pattern as the one observed on the right side of figure 4 - training time grows linearly in the number of training flows, while false positive and false negative rate stay zero.

From the figure we can see that end-to-end training time is linear in the number of flows used in training - with 1,000 unsuspicious flows it is 18 seconds and with 2,000 unsuspicious flows it increases to 34 seconds. Meanwhile, the false positive rate and false negative rates decrease as the number

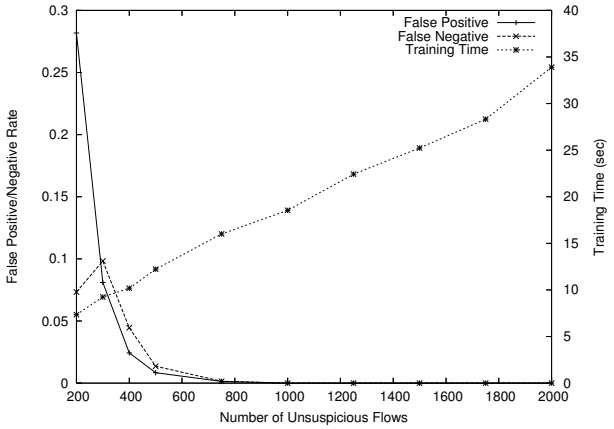

Figure 4. False positive rate, false negative rate and training time under varying number of unsuspicious flows.

of unsuspicious flows grows. When the number of unsuspicious flows is 200, the same as the number of suspicious flows, RIPPER mislabeled 28% of testing innocuous flows as worms and 7.5% of testing worm flows as innoucuous. However, when there are 1,000 or more unsuspicious flows in training, both rates stay zero consistently. This result suggests that we can safely bring the number of unsuspicious flows down to 1,000 and thus reduce the end-to-end training time to 18 seconds.

5. Conclusions and future work

We verified in this paper that certain machine learning algorithms work well for the problem of worm fingerprinting. In particular, we compared the performance of five machine learning algorithms against the best existing worm fingerprinting algorithm (Polygraph) on a blend of network traces and simulated polymorphic worm flows. Results showed that two machine learning algorithms perform significantly better than Polygraph in terms of resilience to noise and detection speed. More specifically, RIPPER produced zero negative rates consistently on noisy training data and was able to capture new worms with very few worm instances in the suspicious pool. Moreover, the algorithm runs in time linear in the total number of training flows, which makes it tractable for containing a large-scale worm outbreak. As future work, we plan to test our techniques on worms with even greater polymorphism using the advanced worm construction ideas presented in [9].

References

[1] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[2] W. W. Cohen. Fast effective rule induction. In A. Prieditis and S. Russell, editors, *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA, USA, July 1995. Morgan Kaufmann. http://citeseer.ist.psu.edu/cohen95fast.html.

[3] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[4] P. Domingos and M. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29:103–130, 1997.

[5] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning (ICML-96)*, pages 148–156, Bari, Italy, 1996. Morgan Kaufmann.

[6] D. Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, 1997.

[7] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. In *IEEE Symposium on Security and Privacy 2004*, Oakland, CA, May 2004.

[8] H.-A. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Proceedings of the USENIX Security Symposium*, Aug. 2004.

[9] O. Kolesnikov and W. Lee. Advanced polymorphic worms: Evading IDS by blending in with normal traffic. In *Gatech CC Technical Report GIT-CC-05-09*, 2005.

[10] J. Z. Kolter and M. A. Maloof. Learning to detect malicious executables in the wild. In *KDD '04: Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 470–478, New York, NY, USA, 2004. ACM Press.

[11] A. McCallum and K. Nigam. A comparison of event models for Naive Bayes text classification. In *Proceedings of the AAAI-98 Workshop on Learning for Text Categorization*, 1998. http://citeseer.ist.psu.edu/mccallum98comparison.html.

[12] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weave. The spread of the Sapphire/Slammer worm. In *Proceedings of the IEEE Security and Privacy*, July 2003.

[13] D. Moore, C. Shannon, and J. Brown. Code-Red: a case study on the spread and victims of an Internet worm. In *Proceedings of the ACM Internet Measurement Workshop*, Nov. 2002.

[14] D. Moore, C. Shannon, G. Voelker, M. G., and S. Savage. Internet quarantine: Requirements for containing self-propagating code. In *Proceedings of the IEEE INFOCOM*, Mar. 2003.

[15] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2005.

[16] V. Paxson. Bro: A system for detecting network intruders in real-time. In *Computer Networks*, Dec. 1999.

[17] T. S. Project. Snort, the open-source network intrusion detection system. http://www.snort.org.

[18] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[19] M. O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Harvard Aiken Computation Laboratory, 1981.

[20] M. Sedalo. Jempiscodes: Polymorphic shellcode generator, 2003. http://www.securitylab.ru/tools/51426.html.

[21] L. Spitzner. *Honeypots: Tracking Hackers*. Addison-Wesley, 2002.

[22] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. A taxonomy of computer worms. In *WORM '03: Proceedings of the 2003 ACM workshop on Rapid malcode*, pages 11–18, New York, NY, USA, 2003. ACM Press.

[23] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufman Pub., San Francisco, 1999.