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 78712, 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 (RIPPER), 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.**

I. Introduction and Background Work

A typical intrusion detection system monitors all the incoming and outgoing traffic, while removing traffic flows that match predefined rules (signatures). However, worm fingerprinting currently requires security experts to manually analyze captured worm instances and thus can be very slow. 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. Moreover, worms released in the past few years have become even more powerful by using polymorphic techniques to avoid detection. As a result, in order to effectively stop worm outbreaks, new automated and robust worm fingerprinting techniques need to be developed.

Among the first *content-based* worm fingerprinting systems, Autograph [1] uses a predetermined heuristic to pre-classify input flows into suspicious and unsuspicious flows, which are then fed as training data to a Rabin's fingerprint based feature extractor and greedy signature generating algorithm.

In Polygraph [2] Newsome et al. propose three algorithms which focus on detecting and generating signatures for polymorphic worms: sequential signatures, conjunctive signatures and Bayesian signatures. Polygraph employs a common substring finder instead of Rabin's fingerprint algorithm to construct features.

II. Worm fingerprinting via machine learning

The task of worm fingerprinting can be abstracted to the problem of constructing a classifier to separate a specific type of flow (worms) from all other flows (innocuous flows) based on their content. There is a range of classification algorithms in the machine learning literature that optimize for classification accuracy – the percentage of instances that are correctly classified – as well as 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.

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 included Naive Bayes learners (NaiveBayes and SparseNB), Support Vector Machines (SVM), Decision Trees (J48), and Rule learners (JRip). These learners were all used "right out of the box" from the Weka data-mining package [3], except for sequential-signature Polygraph (Seq-Poly), which we implemented following the best-performing method from [2].

III. Experimental results

A. Experimental Design

Our experimental comparisons were conducted on a combination of network traffic traces and self-generated polymorphic worm instances. The two network traces – referred to here as the day trace and the week trace – were collected from a 100Mbps fiber link at ICSI, recorded over the span of one day and one week respectively. 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; 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 for these worms were sampled uniformly from the pool of previously constructed innocuous flows, and bodies 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 by employing one of two feature construction techniques: a common substring finder like that used in [2] (COM) and an n-gram finder like that used in [4] (n-GRAM). As in Polygraph, COM looks for all substrings within a predetermined length limit that appear in more than a given percentage of flows. Following [4], 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 suspicious and unsuspicious flows. In order to find the best parameters for the two methods, we conducted development experiments on the day trace.

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 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. To ensure the reliability of the results, for each setting we report the results averaged over ten runs.

B. Accuracy of Generated Signatures

1) With a worm-free unsuspicious pool.: Following the experimental design used to test Polygraph, for different suspicious pool sizes from the week trace, we generated a noise curve for different levels of classification noise in the suspicious pool. From the experiments, we observed that Polygraph started to generate false positives when the level of noise in the suspicious pool increased, which agrees with the results presented in [2].

JRip (the Weka implementation of RIPPER) performed the best among the contending algorithms, as it achieved zero false positive rates consistently; in fact, for most runs, JRip just produced single rules that directly encoded the address of the security flaw in the server system, required for any worm to break into it (and not exploited in innocuous flow payloads). In addition, JRip successfully generated such signatures 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.

2) When the unsuspicious pool contains worms.: The authors of Polygraph make the assumption that the unsuspicious pool is free of worm flows, in order to use flows from a few days earlier to form pure unsuspicious pools. To verify the conjecture that Polygraph will be rendered ineffective if this assumption is relaxed, we repeated our previous experimental setting, additionally blending in twenty simulated worm flows into the unsuspicious pool.

As shown in Figure 1, Polygraph had a consistently high false negative rate, while JRip generated zero false negative rates even 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.

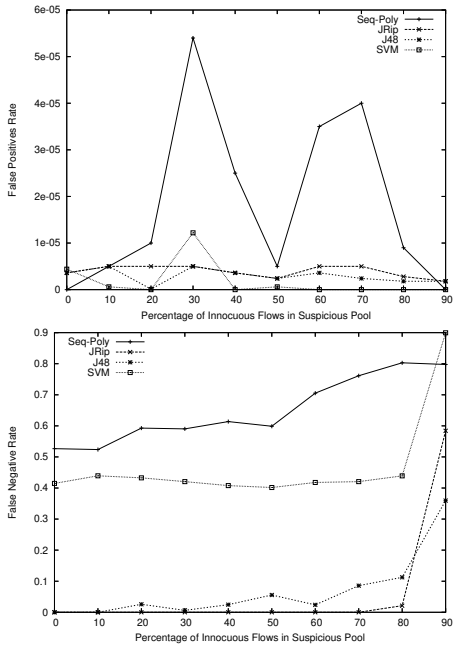

Fig. 1. False positive and false negative rates under varying percentage of noise in the suspicious pool (200 flows). Unsuspicious pool (45,000 flows) contains 20 worm flows.

Worm authors can disable worm detection algorithms by slow-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 worm fingerprinting algorithms to generate erroneous worm signatures for new legitimate innocuous flows (with different characteristic payloads) right after their release, because these flows only exist in the suspicious pool and not in the unsuspicious pool.

C. Training Time

1) Training time for the accuracy experiments.: Figure 2 presents training times for the two experiments presented in the previous subsection.

The time complexity for sequential Polygraph is $O(n^2m)$, where n and m are the numbers of suspicious and unsuspicious flows respectively. This, we believe, 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.The training time of J48 lies between that of JRip and Polygraph.

2) End-to-end training time for production use.: Given the training time of JRip (3 to 10 minutes), we wanted to verify that our approach can be made fast enough to effectively quarantine a worm outbreak. With a more efficient C-implementation of RIPPER, n-grams as feature extractors, and

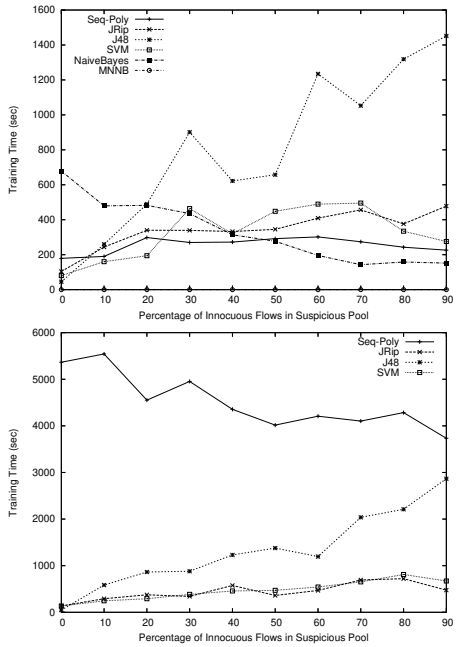

Fig. 2. Training time under varying percentage of noise in suspicious pool. The above graph depicts suspicious pool of size 50 and pure unsuspicious pool. The bottom graph depicts suspicious pool of size 200 and unsuspicious pool with 20 worm flows

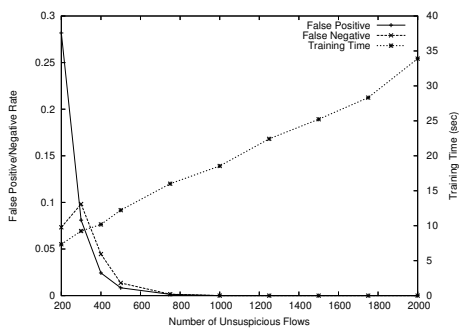

Fig. 3. False positive rate, false negative rate and training time under varying number of unsuspicious flows.

a more streamlined process, we conducted additional experiments to explore how fast the Ripper algorithm could train on a minimum number of examples necessary to identify a worm. We then measured the end-to-end time required to fingerprint a new worm with this "production level" implementation of our approach.

From Figure 3 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. Moreover, when there are 1,000 or more unsuspicious flows in training, both false positive and false negative rates stay at 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.

D. Introducing a few purer labeled flows

Part of our ongoing research involves replaying randomly sampled traffic flows on virtual hosts to see if they are worms. The virtual hosts are equipped with the latest server software, thus a worm flow, when replayed on a host, will reveal its malicious nature by exploiting flaws in the software. This approach, when compared with the suspicious flow capturing algorithms introduced in Autograph and Polygraph, can obtain suspicious and unsuspicious flows that are much purer in nature, but is prohibitive due to the cost of establishing and maintaining the replay engine, as well as the time needed to replay each flow.

Hence, we propose to augment existing fingerprinting algorithms by taking the fewer but purer labeled flows into consideration. One way to incorporate the new flows is to give them larger weights in training compared to the weights given to the less pure suspicious flows. We conducted the same experiments as those done in section III-C.2, while introducing a set of 10 labeled flows that were all worms. These flows were added to the original training set but were given a weight of 5.0 instead of 1.0. The results indicated that the minimum number of unsuspicious flows to ensure zero false positive/negative rates was lowered from 1,000 to 750, which in turn reduced the minimum end-to-end training time by 11.1% down to 16 seconds.

IV. 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 more advanced worm construction ideas.

References

[1] Kim, H.A., Karp, B.: Autograph: Toward automated, distributed worm signature detection. In: Proceedings of the USENIX Security Symposium. (2004)

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

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

[4] Kolter, J.Z., Maloof, M.A.: 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, New York, NY, USA, ACM Press (2004) 470–478