2008

Limits of Learning-Based Signature Generation with Adversaries

Shobha Venkataraman
Carnegie Mellon University

Avrim Blum
Carnegie Mellon University

Dawn Song
University of California - Berkeley

Limits of Learning-based Signature Generation with Adversaries

Shobha Venkataraman
Carnegie Mellon University
shobha@cs.cmu.edu

Avrim Blum
Carnegie Mellon University
avrim@cs.cmu.edu

Dawn Song
University of California, Berkeley
dawnsong@cs.berkeley.edu

Abstract

Automatic signature generation is necessary because there may often be little time between the discovery of a vulnerability, and exploits developed to target the vulnerability. Much research effort has focused on pattern-extraction techniques to generate signatures. These have included techniques that look for a single large invariant substring of the byte sequences, as well as techniques that look for many short invariant substrings. Pattern-extraction techniques are attractive because signatures can be generated and matched efficiently, and earlier work has shown the existence of invariants in exploits.

In this paper, we show fundamental limits on the accuracy of pattern-extraction algorithms for signature-generation in an adversarial setting. We formulate a framework that allows a unified analysis of these algorithms, and prove lower bounds on the number of mistakes any pattern-extraction learning algorithm must make under common assumptions, by showing how to adapt results from learning theory. While previous work has targeted specific algorithms, our work generalizes these attacks through theoretical analysis to any algorithm with similar assumptions, not just the techniques developed so far. We also analyze when pattern-extraction algorithms may work, by showing conditions under which these lower bounds are weakened. Our results are applicable to other kinds of signature-generation algorithms as well, those that use properties of the exploit that can be manipulated.

1 Introduction

It is well-known that automatic signature generation is important – often, there may be small time-windows between when a vulnerability is discovered, and when fast-spreading exploits that target it appear. Generating signatures manually is slow and error-prone, and thus, we need automatic signature generation.

However, there are some requirements for the generated signatures to be useful. These signatures need to identify most of the exploits (have low false negatives), and falsely identify very few non-exploits (have low false positives). They need to capture properties of exploits that do not appear in normal traffic. They also need to be efficient so that signatures can be generated quickly. These requirements make automatic signature generation a hard problem.

A major line of research effort has focused on finding signatures using *pattern*-based analysis, *i.e.*, by extracting byte patterns that uniquely distinguish exploits using network traffic statistics [17, 18, 20, 25, 29]. Such *pattern-extraction algorithms* are attractive because the signatures can be efficiently generated and matched. Pattern-extraction algorithms are, at core, machine learning algorithms: they use a pool of data containing exploits and normal traffic (called the *training pool*), and look for *invariant* byte strings that are present across all exploit packets, but do not occur in the normal traffic. Earlier work has shown that such distinguishing invariants exist, even when the payloads are self-encrypting, e.g., even in polymorphic worms, the high-order bits of the return address of buffer overflows and protocol framing bytes are found to be invariant [20, 25]. This research has led to interest in how pattern-extraction algorithms could be attacked or evaded [15, 26, 28].

In this work, we show fundamental limits on the accuracy of a large class of pattern-extraction algorithms in an adversarial setting. We formulate a framework that allows unified analysis of all such pattern-extraction algorithms, and show lower bounds on the mistakes all pattern-extraction algorithms need to make under some common assumptions, by showing how to adapt results from learning theory. At a high level, our results show that algorithms for pattern-extraction signature-generation can be forced into making a significant number of false positives or false negatives. Earlier work on the limitations of pattern-extraction algorithms have focused on individual algorithms and specific systems. For example, Perdisci et al. [28] demonstrate if an attacker can systematically inject noise into the training pool, Polygraph fails to generate good signatures. Newsome et al. [26] illustrate similar results in Paragraph even when adversary cannot inject arbitrary noise into the training pool. Our results generalize these earlier results through

theoretical analysis, demonstrating that similar attacks are possible on all such algorithms, with similar assumptions.

The central conclusion of our theoretical analysis is that any pattern-extraction (and similar learning-based) algorithms could be manipulated into making a number of mistakes on arbitrary exploits, as a function of the adversary's power to add misleading information to his exploits. Because we cannot predict how future exploits would look, it is important to know (and this result shows us) when and how much pattern-extraction algorithms could be fooled. These results hold when there is a valid signature of invariants, the signature-generator uses randomized algorithms, whose output the adversary cannot predict, and even if host-monitoring techniques like taint analysis [7, 11, 27, 30] are used to identify exactly which packets are exploits and which are not.

Our results are independent of the kind of function that the algorithm tries to learn over the byte sequence, (i.e., the algorithm is allowed to learn any arbitrary complex function over the invariant bytes), or computational complexity of the algorithm. Our analysis also offers insight into algorithms that refuse to tolerate one-sided error, and the lower bounds for these algorithms are much higher than results for more general algorithms. These results show that, it is indeed much easier for an adversary to manipulate an algorithm that makes very few false positives, or very few false negatives.

Existing experimental results (Perdisci et al. [28] and Paragraph [26]) already illustrate that the assumptions for our analysis hold, at least for current families of pattern-extraction algorithms. Our results demonstrate that if pattern-extraction algorithms (and similar signature-generation algorithms) need to work in an adversarial environment, they need to be designed so that the assumptions do not hold; i.e., that the adversary cannot find a large set of byte strings, resemble the exploit's invariants in traffic frequency statistics.

We also explore when pattern-extraction algorithms (and similar signature-generation algorithms) may work. For example, if an exploit contains invariants that are never present in normal traffic, then it seems likely that the exploit can be identified. Our results show that the lower bounds of some families of algorithms are noticeably weakened, under some conditions, *i.e.*, when there is a gap between the distribution of tokens in normal traffic and the invariants of exploits. Our analysis also offers insight into the kind of algorithms that may work and highlights the importance of the function (over the invariants) that the algorithm learns: algorithms that look for a simple set of invariants (learning a simple conjunction of invariants) have far worse lower bounds than algorithms that look for more complex functions over the invariants; this implies that, unlike the results for arbitrary exploits, it is far easier for the adversary to manipulate those simple functions when there is a gap in the traffic.

Our results are also applicable to other signature-generation techniques besides pattern-extraction algorithms. If, for example, a signature-generation algorithm looks for protocol fields exceeding specific lengths, but chooses the lengths based on malicious traffic (e.g., COVERS [21]), our results would still hold. The key limitation in pattern-extraction algorithms is that the adversary can easily add patterns that are similar to exploit's invariants, and algorithm cannot distinguish between the invariants and those added by the adversary (red herrings). Our results are applicable as long as this kind of limitation holds: the adversary can embed similar properties to those invariant to the exploit, and the algorithm cannot distinguish between them.

2 Definitions and Overview

We now present the main definitions and assumptions that we use throughout the paper.

2.1 Definitions

A *signature* is a function σ that classifies a given byte sequence (or, equivalently, packet) as malicious or non-malicious, *i.e.*, $\sigma(y)$ = *Malicious*, when byte sequence y is an exploit, and $\sigma(y)$ = *Non-Malicious*, when y is benign.

A signature may be based on various properties of the byte sequence, and we denote these properties under consideration for signature generation as *attributes*. An attribute A is a function whose input is the byte sequence and output is boolean: e.g., A could be whether $aaaaa$ is present in a byte sequence. For this attribute, for byte sequence $aaaaattttt$, $A(aaaaattttt) = true$, while for byte sequence $aabbccttttt$, $A(aabbccttttt) = false$. A signature could then be considered a function of attributes; thus, in effect, a signature is a function over boolean conditions. e.g., if $A(y)$ and $B(y)$ are attributes, a signature could be $\sigma(y) = \{Malicious : A(y) \wedge B(y)\}$. We say that an attribute is *satisfied* if it evaluates to true.

Recall pattern-extraction algorithms look for invariant byte strings that are present in the exploit, and not in normal traffic, and these invariants are byte strings that must all be present in the exploit for it to be malicious. e.g., a signature reported by Polygraph [25] is a pair of byte strings, '\xFF\xBF' and '\x00\x00\FA' (the Lion worm exploiting the BIND TSIG vulnerability). We refer to each such byte string as a *token*. In our terminology, the attributes test for whether each of these tokens is present, and signature is a conjunction of the two attributes. We could denote this signature as $\sigma(y) = \{Malicious :$ '\xFF\xBF'$\in y \wedge$ '\x00\x00\FA' $\in y\}$.

More generally, for pattern-extraction algorithms, an attribute tests for the presence of a particular token, perhaps at a particular location. For algorithms that use more information, other kinds of attributes would also be needed, e.g., COVERS [21] considers lengths of fields, so an attribute would also represent whether a particular field in the byte sequence is longer than a specific value.

For a fixed set of attributes G, we can represent a byte sequence by the attributes in G that it satisfies, and we describe how to do so now. We define an *instance* i for a byte sequence and a set of attributes G to be a boolean m-tuple, i.e., $i \in \{0,1\}^m$, where the ith bit is 1 if the ith property holds true for the byte sequence. An instance thus is a representation of a byte sequence for a set of attributes. So, if G consists of the two attributes "Is $\{aaaaa\}$ present?" and "Is $\{bbbbb\}$ present?", the byte sequence $aaaaaxxxxbbbb$ would be represented as $(1,1)$, and $ccccxxxxxbbbb$ would be represented by $(0,1)$. The *instance space* is the set of all instances $I = \{i \in \{0,1\}^m\}$. For the rest of this paper, we consider the set of possible attributes G to be fixed. Every byte sequence is represented as a vector in the instance space, and our discussion will be in this instance space $\{0,1\}^m$.

We also introduce some machine learning terminology. The algorithm is given some *training* data, which consists of malicious and non-malicious instances, along with the *labels* of each instance, so that it knows which instances are malicious and which are not. The algorithm finds a *hypothesis*, a function that classifies a given instance as malicious or non-malicious.

We define the *true signature* to be the signature that achieves 0 false positives and 0 false negatives, on any set of instances presented to it. In learning terminology, the true signature is the *target hypothesis* that needs to be found by the learning algorithm. Once the space of attributes is fixed, we assume there is only one true signature; of course, there may be multiple functions to represent this true signature.(If there are multiple signatures that can always achieve 0 false positives and false negatives, a conjunction of these signatures is also a true signature, so this assumption is not limiting.) We refer to the attributes in the true signature as *critical attributes*.

2.2 Overview of Learning Framework

The central question that we want to answer is the following: to what extent can an adversary force every learning algorithm to learn the signature slowly? We answer this question by presenting lower bounds on the algorithm's performance. To do so, we need some assumptions, and in this section we describe and justify some basic assumptions we use. Our goal in choosing these assumptions is to give the algorithms in the signature generator as much power as possible. The lower bounds show that, even so, the adversary can evade detection for a long time.

We focus on four different assumptions here: the learning model, the form of the true signature, the label correctness, and the adversary's knowledge of the algorithms.

Learning model: Our analysis assumes that the algorithm is allowed to update its internal state after each batch of data that it sees, and these updates may be made over all of the data accumulated so far. For example, if the algorithm has 100 packets in its initial training pool, and then gets 50 packets in the next batch, the algorithm may update its signature after seeing the second batch, and may then use all 150 packets to generate the updated signature.

This is a little different from the typical machine learning setting, where the algorithm is given a large batch of training data, allowed to learn a function over it, and then tested on new testing data. However, since we have a malicious adversary who controls part of the data and aims to delay learning, the adversary could ensure that, without updates, the algorithm never learns a good signature. By allowing updates, algorithm might find a good signature over a longer period of time. We can also perform a more informative analysis about how the algorithm's performance evolves with more data over time. The learning algorithm still does get an initial training pool, which can contain any number of malicious and non-malicious samples, as long as the assumption of Section 3 is obeyed.

In addition, since the adversary wants the algorithm to make as many errors as possible, the adversary aims to release information about the true signature as slowly as possible. The adversary can present information about exactly one new instance in each batch (e.g., all the malicious instances in the batch can be the same, so a mistake on the instance would cause a 100% false negative). [1] In effect, it is as if the algorithm gets one new instance at a time, classifies it, and updates its internal state based on that instance. Our bounds will be in terms of the number of mistakes the algorithm makes in this setting, which also corresponds to the number of updates it requires. In the learning theory literature, this is known as the mistake-bound model [22].

Form of the true signature: We assume that the true signature of the exploit is a conjunction of attributes – i.e., all attributes in the conjunction must be satisfied by the packet. We do so because conjunctions are the simplest form of signatures that have been historically considered, and lower bounds for conjunctions imply lower bounds for more complex functions that can represent conjunctions. For example, these lower bounds are also lower bounds for regular expressions, because regular expression signatures can represent conjunctions.

However, we do not make any assumption on the form

[1] Indeed, if the algorithm can update its hypothesis only every batch, it is optimal for the adversary to present exactly one instance at a time.

of hypotheses chosen by the algorithms for its internal state. The algorithm could use, for example, a weighted combination of tokens as its classifier. The learning algorithm is *not* required to learn a conjunction.

Label Correctness: We assume that every label given to the algorithm is correct. This means the adversary is forced to be truthful, and cannot decide to change the signature (target hypothesis) after the algorithm has been given data. This affords the algorithm a lot of power: it is as if the algorithm has an oracle like a dynamic runtime checker, and can test each input on it. If the adversary can lie, by adding carefully crafted noise for some instances, or change the target, the lower bounds would only increase.

Adversary knowledge: We assume that the adversary knows the kinds of attributes considered by the algorithm in question. In some of our bounds (for deterministic algorithms), the adversary needs to know the algorithm he aims to mistrain, but this is not required for the bounds on the randomized algorithms. These randomized bounds hold when the adversary (a) knows the algorithm, but has no access to its private randomness, (b) does not know the algorithm, or (c) does not know the parameters (e.g., statistical algorithms using soft decision boundaries).

3 Reflecting Set

In this section we describe the formal framework we will use to analyze limitations on learning-based signature generation. Our key assumption is that the adversary has the ability to construct *reflecting sets*: spurious attributes (e.g., tokens) that, to the learning algorithm, look at least as plausible apriori as the actual attributes in the signature. These take the role of the "concept class" in learning theory, and the larger the set, the stronger the limitation. Below, we motivate the notion of reflecting sets and give formal definitions, focusing on pattern-extraction algorithms, especially Polygraph [25] & Hamsa [20].

3.1 Motivation & Definitions

We begin by observing a common property of many strategies proposed to evade detection by pattern-extraction algorithms. A wide range of strategies have been proposed for evasion, and all of them succeed because the adversary can increase the number of tokens that resemble the tokens critical to the signature. e.g., In red herring attacks [25], the attacker adds spurious tokens to the true signature, and the attack succeeds when the algorithm mistakenly considers those as part of the true signature. Likewise, in noise injection attacks [28], allergy attacks [15] and suspicious/innocuous pool poisoning attacks [26], the adversary manipulates the token distribution in the training or testing pool, by adding well-crafted (malicious or normal) packets with carefully chosen tokens, and changing the distributions of various tokens in the training pool. Here again, how effective the attack is depends on how much the attacker can change the tokens considered by the algorithm.

Thus, these attacks succeed when the algorithm is unable to apriori distinguish between the tokens critical to the true signature, and any spurious tokens that happen to resemble these critical tokens. The attacker forces the algorithm to fail by carefully increasing the appropriate resemblance between the critical and spurious tokens, and he may be able to do this for other kinds of attributes as well. We will use the term *reflecting sets* to describe these sets of resembling attributes, as each attribute within a reflecting set appears to reflect all of the other attributes in that set.

Definition: We now define reflecting sets formally: Let S denote the true signature for an exploit, and let A denote a signature generation algorithm. Let $Pr_A[S']$ denote the probability that A gives to the function S' being the true signature. Suppose $C_1, C_2, \ldots C_j$ be sets of attributes, such that the signature S contains an attribute s_i in each C_i. Let $\mathcal{T}$ is be the set of functions obtained by choosing one or more attributes $c_i \in C_i$, to replace the corresponding property s_i in S. Let W_m and W_{nm} be the malicious and non-malicious instances seen by the algorithm so far, and let $W = W_m \cup W_{nm}$. Let $\mathcal{T}_W$ be the set of functions in $\mathcal{T}$ consistent with W. If $Pr_A[T] = Pr_A[T']$, for any pair of functions $T, T' \in \mathcal{T}_W$, for all W, then the sets $C_1, C_2 \ldots C_j$ are *reflecting sets* for the signature S and the algorithm A.

Thus, from the point of the view of the algorithm, it is as if any combination of attributes, as long as one is picked from each reflecting set, could be the true signature even after analysis over all of the training data. If $\mathcal{T}$ denotes the set of all combinations of attributes that includes one from each reflecting set, then *to the algorithm*, the true signature S appears to be drawn at random from $\mathcal{T}$.

An additional aspect of this definition is that reflecting sets are specific to an algorithm (or a family of algorithms). We define the reflecting sets this way because different algorithms could use different aspects of possible attributes to identify a likely signature, and therefore, reflecting set for one algorithm may not be a reflecting set for another algorithm. For example, the conjunctions algorithm of Polygraph uses every infrequent token that appears in all of the malicious instances as its signature. For this algorithm, a reflecting set is very easily constructed by simply adding more infrequent tokens to all of the malicious instances. Such a simple reflecting set, however, would not work for other algorithms, e.g., naive Bayes algorithm in Polygraph, or Hamsa's algorithm.

Learning with Reflecting Sets: In this paper, we analyze the problem of learning a signature with a malicious

adversary as the following: for every critical attribute, the adversary may include the respective reflecting set in the packets (normal or malicious, as needed). The goal is to find the true signature by identifying the critical attributes, isolating them from their reflecting sets. We define the problem formally in the next section.

3.2 Finding Reflecting Sets

The results of this paper are applicable to algorithms where it is possible/easy for the adversary to construct reflecting sets (or sets with a bias away from the true signature) for the attributes in the true signature, e.g., pattern-extraction algorithms. In general, this could be done for algorithms that require information from (adversarially-generated) exploits, but cannot identify the true cause of the exploit, and therefore, the attribute or parameter they learn can be forged by the attacker.

It is also not strictly necessary for all the attributes in the reflecting set to have identical traffic statistics: the goal is to capture the algorithm's inability to distinguish between different attributes inside the set, and therefore, unable to bias any selection towards the true signature. If the reflecting sets are chosen so that the algorithm's choice of signature is *less* likely to be the true signature, then the lower bounds would only increase. e.g., Hamsa's algorithm prefers tokens with the smallest frequency in normal traffic pool, and the attack suggested adds spurious tokens that are even less frequent, and therefore, would cause the algorithm to make at least as many mistakes.

The quantitative bounds on algorithms' errors are related to the size of the reflecting sets that can be found for the attributes. The size of any particular reflecting set depends on the nature of the exploit (e.g., its distinguishing properties, the protocols applicable), the adversary's ability to manipulate the training and testing pool, and the kinds of signatures that the algorithm aims to learn for it. The adversary may craft these reflecting sets either by explicitly including selected attributes in the malicious instances, or sending specific types of instances in the training data. Because the adversary crafts the reflecting set for the signature generators, the adversary knows the reflecting set.

Earlier experimental work (e.g., in Paragraph) has demonstrated that reflecting sets can be found for current generations of pattern-extraction algorithms. Further, polymorphic blending attacks [14] suggest that it may be possible to find such reflecting sets for many pattern-extraction algorithms, as long as the algorithms use byte-based traffic statistics for finding the priors of the critical tokens in the signature (e.g. [33]). We believe it would be typically possible to find reflecting sets for pattern-extraction algorithms in general, especially those which use the traffic statistics of individual tokens, due to the heavy-tailed nature of normal traffic distribution.

4 General Adversarial Model

In this section, we consider a general adversarial setting, and we present impossibility results on learning algorithms that generate signatures in this model.

4.1 Learning Model

We present our analysis in the mistake-bound model of learning. As described in Section 2.2, we choose this model because it affords the algorithm significant power, but even with this power, the adversary can delay signature generation. In this model, the algorithm gets an initial training pool (of any size), and then gets one instance at a time to classify, classifies it as malicious or non-malicious, and is then told the correct label of the instance. The algorithm then updates its hypothesis. The algorithm's goal is to converge to the true signature while minimizing the mistakes made.

Each instance given to the learning algorithm is an m-tuple boolean vector, i.e., a point in $\{0,1\}^m$. The true signature, or target hypothesis, is a conjunction of n attributes: an instance must contain all n attributes to be malicious. As discussed in Section 3, we assume the adversary can find a reflecting set C_i of size k for each critical attribute i, and the algorithm cannot distinguish between the attributes inside C_i. It may, however, be able to distinguish between attributes in different reflecting sets, and we need to account for this in the lower bounds. Thus, the set of all valid hypotheses is the set of all conjunctions containing an attribute from each reflecting set; thus $|H| = k^n$. We refer to the n bits in the true signature as the *target bits*. Because the adversary crafts the malicious data, he can ensure that even with an initial training pool, no information is released about the critical attribute, to distinguish it in its reflecting set. The total number of attributes $m = nk$, the product of the number of critical attributes and the size of each reflecting set.

Our bounds are in terms of the number of mistakes made by the algorithm. The mistakes made can be interpreted as the number of updates required to converge to the true signature, when the algorithm receives the correct label right away. The mistakes in this model imply false positives and negatives in the standard batch setting: a mistake on a malicious instance is a false negative, and a mistake on a non-malicious instance is a false positive. The exact false positive and negative rate that a mistake (or a sequence of mistakes) causes depends on the specific algorithm, but a worst-case estimate on any particular batch can be seen: whenever the algorithm makes a mistake, the adversary can generate a distribution that causes a 100% false negative rate (for a

malicious instance), or potentially a large false positive rate (for a normal instance).

There are two ways in which target hypothesis can be chosen for the lower-bounds analysis. The adversary can choose the target hypothesis from the set H, or nature picks the target at random from the set H, and the adversary knows the target hypothesis selected. Lower bounds for the second way of choosing the target clearly imply lower bounds for the first.

Representation of Hypothesis Even though the target hypothesis is a conjunction of the target bits, there is no requirement that the learning algorithm learn a conjunction of the target bits. That is, the learning algorithm is free to choose any function, as long as it agrees with the target hypothesis on all the instances seen.

Formally, let $x_1, \ldots, x_m$ denote m bits of an instance, where $x_j = \{0, 1\}$. In this context, a conjunction hypothesis is a function $x_a \wedge x_b \wedge \ldots x_r$, for some r values, and evaluates to true if all bits $x_a \ldots x_r$ are 1. A *linear separator* hypothesis is a function of the form $\sum_{i \in [1,m]} w_i x_i > q$ where the weights $w_i \in \Re$. All instances that satisfy the condition (i.e., weighted combinations of bits exceeds the threshold q) evaluate to true.

The *representation* of the hypothesis is the type of function learnt by the algorithm, e.g. a linear separator or conjunction. Polygraph uses both conjunctions and linear separators, and Hamsa uses conjunctions. The results in this section are independent of the hypothesis representation chosen.

4.2 Results

We now present our results in the learning model described above. Each of these lower bounds can be derived from more general results in learning theory; our proofs show an explicit construction of instances that achieve the bounds for our setting. For space reasons, we defer all proofs to the appendix. In the proof of each theorem, we show a sequence of instances for which any algorithm must achieve the stated mistake-bound. [2]

We first present bounds on the overall number of mistakes that any deterministic or randomized algorithm could be forced to make. Theorem 4.1 shows that every deterministic algorithm, regardless of what it learns, could be forced to make at least $n \log \frac{m}{n}$ mistakes by an adversary – thus, the mistakes grow linearly in the size of the signature, but only logarithmically in the size of the reflecting sets.

[2]The adversary does not require knowledge of the algorithm's behaviour to generate the next instance. He would for the lower bounds on deterministic algorithms, but the bounds for the randomized algorithms apply even if the algorithm is a blackbox to the adversary.

Theorem 4.1. (Deterministic Algorithms) *For every deterministic algorithm, an adversary can generate a sequence of instances such that the algorithm is forced to make at least $n \log k$ mistakes, where k is the size of the reflecting sets.*

Since the bound of Theorem 4.1 scales logarithmically with the number of spurious attributes, it is natural to ask whether this lower bound is tight. The Winnow algorithm [22] achieves a bound within $n \log n$ additive factor, showing that the bound is nearly tight.

However, much of the error in the previous theorem comes from the adversary's ability to predict what the algorithm would do next. A common solution is to allow the algorithm to use randomization. Theorem 4.2 analyzes the number of mistakes made if the algorithm is randomized (or equivalently, unknown) to the adversary. It shows that even if the signature generator uses a randomized algorithm, the algorithm can be forced to generate a lot of mistakes in expectation, half the mistakes of the deterministic case.

Theorem 4.2. (Randomized Algorithms) *For any randomized algorithm, an adversary can generate a sequence of instances so that the algorithm will make, in expectation, at least $\frac{1}{2} n \log k$ mistakes, where k is the size of the reflecting sets.*

Theorem 4.2 shows that an arbitrary deterministic algorithm is not too much worse than a randomized algorithm, and suggests that some deterministic algorithms may not fare too poorly. This result is, however, dependent on the nature of determinism in the algorithm. For example, one kind of extreme determinism is to guarantee no false positives or no false negatives. Such algorithms are attractive, since it seems better to have to tolerate only one kind of error.

We now consider one-sided algorithms: algorithms which are not allowed to make (many) false positives or (many) false negatives. Our results show that one-sided algorithms can be forced into making many more errors than algorithms with arbitrary break-down of mistakes (e.g., in comparison to Theorem 4.1). Guaranteeing a small number of mistakes of either false positives or false negatives forces the algorithm to make a large number of mistakes of the other kind.

Theorem 4.3. (Bounded False Positives) *If an algorithm that is not allowed to make any mistakes on non-malicious instances, there exists a sequence of instances such that it is forced to make at least $n(k-1)$ mistakes on malicious instances. More generally, consider an algorithm that is forced to make fewer than t mistakes on the non-malicious instances, for $t \leq n$. Then the algorithm must make at least $(n-t)(k-1)$ mistakes on the malicious instances.*

Such large mistakes are not special to only algorithms that require a small number of false positives. Theorem 4.4 shows mistake-bounds for algorithms that must make very few false negatives. Indeed, these mistake-bounds are much larger than those in Theorem 4.3, for $kn \gg n$ (i.e., since reflecting sets are large but contain only one target bit each).

Theorem 4.4. (Bounded False Negatives) *If an algorithm is allowed to make no mistakes on malicious instances, an adversary can generate a sequence of instances so that the algorithm is forced to make $k^n - 1$ mistakes on non-malicious instances. More generally, consider a deterministic algorithm that is forced to make fewer than t mistakes on malicious instances, for $t < n$. Then the algorithm must make at least $k^{\frac{n}{t+1}} - 1$ mistakes on non-malicious instances.*

We note briefly that lower bounds in Theorems 4.3 and 4.4 are may not be tight for large values of t. Nevertheless, they still serve to illustrate the effect of allowing very few false positives or false negatives.

We note also that the bounds of Theorems 4.3 and 4.4 are very different. Intuitively, the basic difference between them arises from the kind of the information that is encoded in an exploit (malicious instance), when compared to a non-exploit (non-malicious instance) packet. In Theorem 4.3, the adversary forces the algorithm to learn the exploit from only exploit information, while in Theorem 4.4, the adversary forces the algorithm to learn the exploit from only non-exploit information. As there may be far more non-exploit packets than exploits, each of which encodes very little information about the exploit, the adversary can be able to force many more errors in Theorem 4.4.

4.3 Practical Implications

Discussion The central conclusion of the theoretical analysis is that any pattern-extraction (and similar learning-based signature-generation) algorithms could be manipulated into making a significant number of mistakes, in terms of the total number of false positives and false negatives generated. This holds when the signature-generator uses randomized algorithms, whose output the adversary cannot predict. It holds even if host-monitoring techniques like taint analysis [9, 11, 27, 30] are used to identify exactly which packets are malicious and which are not. Our analysis suggests that these algorithms could work only when they are designed so that a large reflecting set cannot be found.

Existing experimental research has already demonstrated the feasibility of these attacks on real systems, e.g. Paragraph shows that it is feasible to add a large number of tokens to a real buffer-overflow exploit against ATPhttp web server, and shows how this affects the detection of polymorphic worms by Polygraph and Hamsa. This disruption

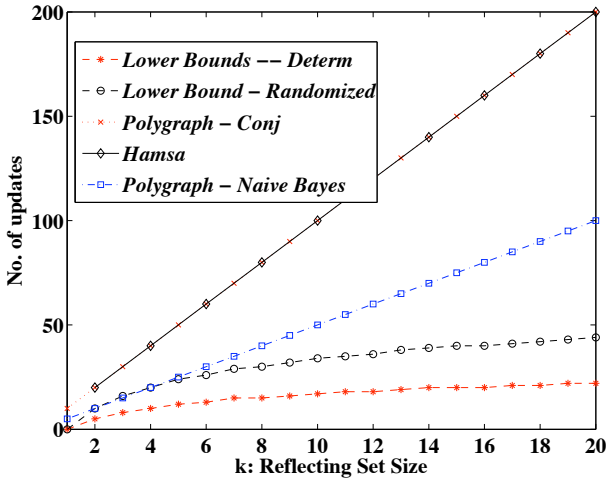

Figure 1. Comparison of lower bounds and current algorithms for general case: number of updates required before convergence to the true signature.

is caused by the sequence of the instances presented to the algorithms, so that the algorithm does not have enough information to infer the correct target. In our proofs, we show constructions of sequences of instances that force every algorithm to make a lot of mistakes.

The results also show that if pattern-extraction algorithms need to be used, an algorithms like Winnow [22] may guarantee better accuracy in adversarial settings. Using a more complex algorithm would not gain a significant improvement. This is especially so these bounds are independent of the representation of the algorithms used (one could learn any arbitrary complex function over the instance space), or the algorithm's computational complexity. Still, Winnow's mistake-bound offers insight into using a more expressive representation than the basic conjunction used in several algorithms.

Lastly, our analysis offers insight into algorithms that refuse to tolerate one-sided error. The mistake bounds for these results are much higher than results for more general algorithms. These results show that, it is indeed much easier for an adversary to manipulate an algorithm that makes very few false positives, or very few false negatives. Specifically, note differences in their dependence on k, the size of the reflecting set: Theorems 4.1 and 4.2 have a logarithmic dependence on k, while Theorems 4.3 and 4.4 have a polynomial dependence on k. Thus, for an arbitrary algorithm, the adversary would not gain significantly from arbitrary increases to the padding, though he does so for one sided algorithms.

Comparison to Existing Systems As a specific illustration of the bounds, we compare lower bounds with the estimated mistakes (through calculation) that would be made by the Polygraph suite of algorithms and Hamsa. These mistakes are made when reflecting sets can be found for these pattern-extraction algorithms, but this has already been demonstrated in Paragraph.

For our comparisons, we use the attacks suggested in Paragraph for each of the algorithms. In the red herring attack suggested on Polygraph's conjunction algorithm, a mistake can be induced on every presented instance by dropping a token each time. Likewise, for the Hamsa's algorithm, a mistake can be induced for each spurious token by dropping the token with the smallest frequency at the time. On the naive Bayes algorithm, mistakes can be induced by the correlated outlier attacks shown: each malicious instance presented is crafted with tokens appearing in normal traffic, forcing the algorithm to classify it as non-malicious.

Fig. 1 shows the number of iterations in which mistakes would be made, as a function of the size of the reflecting set, for a signature with 10 tokens. Each of these algorithms require iterations linear in nk, where n is the number of true critical attributes in the signature, and k is the size of the reflecting set. The lower bounds, on the other hand, grow logarithmically in k. Of course, it is harder to find a reflecting set for the Bayes algorithm than the conjunction algorithm in Polygraph, and it is similarly harder to find a reflecting set for Hamsa's algorithm. For example, if there are 10 tokens in the true target signature, a reflecting set of size 10 for each token would mean that the lower bounds for deterministic algorithms require 34 updates, and the randomized algorithms require 17 updates. In contrast, the Polygraph conjunction algorithm and Hamsa's algorithm could be manipulated into requiring 100 updates.

Further, all of these calculations assume that there is an effective way to ensure that the presented instances are (later, at time of update) correctly classified, and that updates are immediate. A lag in updates would increase the number of batches seen (and therefore, mistakes seen) before converging. If, for example, the algorithm gets the correct label only every 10 iterations, the number of mistakes could increase by a factor of 10.

5 Exploiting Gaps in Traffic

In this section, we examine when signature-generation algorithms would work, even in the presence of adversaries, and when there may be large reflecting sets for the signatures. For example, if an exploit's invariant tokens *never* appeared in normal traffic, it ought to be possible to identify this exploit with pattern-extraction algorithms. Our goal is to understand conditions under which these learning algorithms might work, even if they must learn over properties which have reflecting sets.

The lower bounds analysis in the previous section was based on the existence of a sequence of instances (or equivalently, an adversary who generated a sequence of instances) for the algorithm to classify, and these instances could be drawn from any point in the instance space. Thus, effectively, the algorithm needed to be able to classify every single instance correctly, and was required to have a small number of mistakes on any sequence of instances. The hypothesis found by the algorithm was required to agree with the target hypothesis on every single instance in the instance space.

However, there might be situations when such a requirement is more stringent than necessary. For example, while we would certainly want the algorithm to be able to classify all malicious instances generated by the adversary, perhaps we do not need the algorithm to classify all possible non-malicious instances, unless they are regularly present in normal traffic. A reasonable goal might be to ask an algorithm to only classify correctly the non-malicious instances that are truly present in normal traffic, rather than any arbitrary combination of properties generated by an adversary. In this situation, an algorithm would need to agree with the target hypothesis only on the malicious instances, and the non-malicious instances *present in normal traffic*.

Thus, algorithm can disagree with the target hypothesis on a region of the instance space, the region where the non-malicious instances are not present in normal traffic. In this case, it might be possible to make fewer mistakes, as a function of how large the gap between the malicious instances and the normal instances are. The analysis in Sec. 4 addresses the case when there is no gap between normal instances and malicious instances.

Recall that the instance space is a boolean hypercube in $\{0,1\}^m$. The malicious instances are instances which have all n target bits set to 1, regardless of the values of the remaining $m-n$ bits. The non-malicious instances are all the remaining instances. The non-malicious instances truly present in normal traffic may be only a small subset of these instances. We need a way to quantify the region of the instance space that the algorithm does not need to classify correctly. We do this by defining how to measure the gap between the two types of traffic in Section 5.1. Then, in Section 5.2, we describe the learning model. We describe results in Section 5.3, and their practical implications in Section 5.4.

5.1 Defining the Gap

Intuitively, our goal is to measure how close the normal traffic is to the malicious instances, e.g., if few attributes of the malicious instances are present in the normal instances,

we would like the gap to be large. Further, we would like the gap to capture some intrinsic property between the normal traffic and the malicious instances, which the adversary cannot manipulate over time. That way, we can then measure the effect of the adversary's manipulation of the malicious instances for different kinds of gap.

We measure the gap in the following manner: let Z be the set of target attributes – the attributes that must truly be present in the malicious instances (in our notation $n = |Z|$). We define the *instance-overlap* of a normal instance i to be the fraction of attributes of Z that is present in the instance. We define *overlap-ratio* of the normal traffic to be the maximum instance-overlap of *any* instance in normal traffic. In other words, the fraction of target attributes present in a normal instance is, at most, the overlap-ratio. So, for example, an exploit whose invariant is a single token that never appears in normal traffic has overlap-ratio 0. Our definition is motivated by the observation that because tokens extracted in signatures are very rare in normal traffic, and the appearance of multiple tokens together is even rarer.

5.2 Learning Model

The learning model in this section is similar to the one in Section 4, however, we need to make some crucial changes. A hypothesis is *overlap-equivalent* to the target hypothesis if the two hypotheses agree on all the malicious instances, and all non-malicious instances truly present in the normal traffic. The goal of the learning algorithm is to find an overlap-equivalent target hypothesis, when the target hypothesis is drawn at random from the set of all valid hypotheses. As in Section 4, we give mistake bounds for algorithms that are allowed any number of samples, and any kind of running time. However, the bounds now depend on the representation that the algorithm uses to find an overlap-equivalent hypothesis.

We use d to denote the overlap-ratio of the normal traffic distribution with the target hypothesis. The overlap-ratio also has an implication for the reflecting sets that the adversary chooses. The attributes in the reflecting sets may also need to obey the overlap-ratio, otherwise, they may not be reflecting sets for some algorithms anymore. That is, these sets may need to be chosen so that no more than d fraction of the reflected attributes from different reflecting sets can be present together in any instance in normal traffic.

5.3 Results

We now present lower bounds on the mistakes made in this model. Unlike the previous model, these results depend on the representation used by the algorithm, whenever the overlap-ratio $d < 1$. This is because there is always a signature that can be represented in the disjunctive normal form: the signature just looks for the presence of *any* of the k^n possible combinations of the attributes is always correct. [3] To our knowledge, this model has not been analyzed before. We leave all proofs to Appendix B.

We describe lower bounds for two commonly used representations: conjunctions and linear combinations of attributes, and we show lower bounds on both deterministic and randomized algorithms. Our bounds are in terms of the mistakes made on a sequence of instances *consistent* with a given overlap-ratio d: every non-malicious instance in the sequence has an instance-overlap of at most d. As these instances are consistent with overlap-ratio d, they could potentially appear in normal traffic. In other words, our theorems imply that, when the overlap-ratio is d, there exists normal traffic for which every algorithm has to make a certain number of errors (as a function of d).

Theorems 5.1 and 5.2 show lower bounds for learning conjunctions for deterministic and randomized algorithms. They show that the mistakes made by any algorithm that is forced to learn conjunctions of attributes scales linearly with the number of attributes, as well as the overlap-ratio of the normal traffic distribution.

Theorem 5.1. (Deterministic Algorithms using Conjunctions) *Let the overlap-ratio of the normal traffic be d, and let k be the number of attributes in each reflecting set. For any d, there exists a sequence of instances consistent with overlap-ratio d such that any deterministic algorithm that learns an overlap-equivalent conjunction will need to make at least $(k-1)(dn+1)$ mistakes.*

Theorem 5.2. (Randomized Algorithms using Conjunctions) *Let the overlap-ratio of the normal traffic be d, and let k be the number of attributes in each reflecting set. For any d, there exists a sequence of instances consistent with overlap-ratio d such that any randomized algorithm that learn an overlap-equivalent conjunction will make, in expectation, at least $\frac{k-1}{k}(dn+1)$ mistakes.*

Next we consider the minimum number of malicious instances that an adversary can send through undetected, if the learning algorithm learns linear separators. Theorems 5.3 and 5.4 show lower bounds for algorithms that need to learn overlap-equivalent linear separators.

Theorem 5.3. (Deterministic Algorithms using Linear Separators) *Let the overlap-ratio of the normal traffic be d, and let k be the number of attributes in each reflecting set. For any d, there exists a sequence of instances consistent with overlap-ratio d such that any deterministic algorithm that learns overlap-equivalent linear separators will need to make at least $\log_{1/d} k$ mistakes.*

[3] Alternately, one can consider this signature to be an OR function of the set of all valid hypotheses described in Section 4.1.

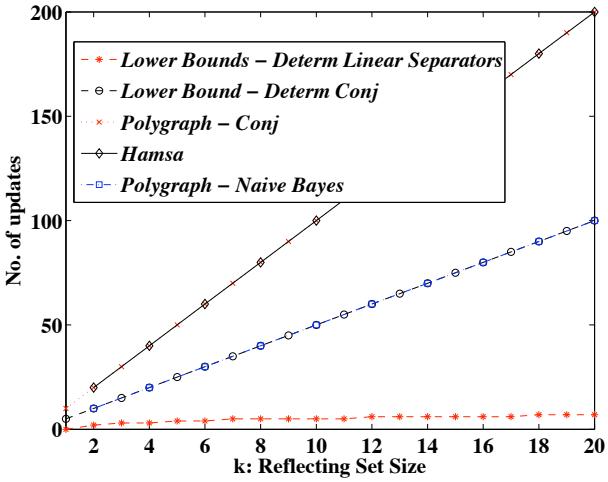

Figure 2. Comparison of lower bounds and algorithms when there is a large gap between the normal traffic and malicious samples: no. of updates required before converging to true signature, as a function of reflecting set

Theorem 5.4. (Randomized Algorithms using Linear Separators) *Let the overlap-ratio of the normal traffic be d, and let k be the number of attributes in each reflecting set. For any d, there exists a sequence of instances consistent with overlap-ratio d such that any randomized algorithm that learns overlap-equivalent linear separators will need to make, in expectation, at least $\frac{1}{2}\log_{1/2d} k$ mistakes to converge to a hypothesis equivalent to the target.*

Since these lower bounds are representation-dependent, they cannot be directly compared to the ones in Section 4.2. However, the results for learning overlap-equivalent linear separators are comparable to the lower bounds of Theorems 4.1 and 4.2: we know that the lower bounds of Theorems 4.1 and 4.2 are tight, and the Winnow algorithm learns a linear separator. We note that $d = \frac{n}{n-1}$, these lower bounds approach those of Section 4.2. Thus, when the gap between the normal traffic and malicious exploits are large, it may be possible to learn with few mistakes.

5.4 Practical Implications

The results of this section suggest that pattern-extraction (and similar signature-generation) algorithms would work in practice for some kinds of exploits – they would work better when the overlap between tokens present in normal traffic and exploits is large. Our analysis suggests an easy way of quickly quantifying the exploits such algorithms may work well for. In addition, it highlights the importance of choosing an appropriate representation to learn from: even if all signatures are conjunctions of tokens (attributes), choosing a more flexible representation like linear separators allows the adversary fewer ways to manipulate the algorithm's behaviour.

Fig. 2 also shows that the representations chosen by the learning algorithm determine its accuracy significantly. It shows the number of updates required to learn the true signature, when there are 10 tokens in the true signature, and the overlap-ratio is 0.5 (i.e., any normal instance has at most half the critical tokens). When the reflecting set is 10, algorithms learning conjunctions still require 50 mistakes, while those learning linear separators require only 5. Conjunctions are easy for an adversary to manipulate, and therefore can be forced to make far many more errors than linear separators. The errors on linear separators also illustrate the extent to which the bounds are weakened with a gap in traffic: the corresponding mistake-bound for arbitrary exploits is about 30.

Of course, normal traffic is difficult to model and may undergo rapid changes. It may be difficult to tell what the overlap-ratio of an exploit might be, and how likely it is to change, and of course, one cannot predict the overlap-ratios of future exploits. Further, the data captured to find the overlap-ratio might not be sufficient to identify a very rare token, and one might think that the overlap-ratio is smaller than it truly is, which would cause false positives. However, if normal traffic continues to originate from the same kind of distribution as the data captured, such false positives are likely to be few and infrequent.

6 Related Work

As we have discussed pattern-extraction signature-generation algorithms throughout this paper, we do not discuss them further here. Signature generation algorithms that use semantic information have taken many different directions; some examples to point at directions are [4, 8, 12, 19, 31]. As it is not immediately clear when reflecting sets would exist if semantic information is used, our results may not apply to these algorithms. A notable exception is COVERS [21], that uses protocol semantics, but generates a property that can be manipulated by the adversary.

We next discuss prior attacks on pattern-extraction algorithms. Perdisci et al. [28] showed that if the adversary could add malicious noise to suspicious pool and the normal pool, Polygraph fails to generate good signatures. Paragraph [26], demonstrates that even with a truthful adversary, Polygraph and Hamsa [20] are vulnerable to attacks. Allergy attacks, forcing many false positives and DoS against the network, also demonstrated on Polygraph and Hamsa [6]. However, these papers demonstrate attacks

on specific algorithms and systems, while our work shows general lower bounds. Gundy et al [15] present a different kind of attack showing that polymorphic worms do not need to have invariant bytes. Our work differs as it shows lower bounds even when there invariant bytes. A related attack on intrusion detection systems are the polymorphic blending attacks by Fogla et al. [14]. These attacks match all byte frequency statistics of normal traffic under consideration by an IDS, and thus evade detection. This is different from our situation, as we do already have the appropriate target attributes under consideration, and these do uniquely identify the exploit. Our work is also complementary to that of Crandall et al [10], as their work explores the extent to which pattern-based signatures may need to be present at all in the packets containing exploits. Our work shows that even if they are present, it is quite easy for the adversary to mislead signature generators.

Finally, we discuss related work in learning in adversarial settings. The learning theory community has explored theoretical questions on learning with malicious adversaries and malicious noise. [2, 5, 16] In this regard, the most related work is mentioned in Sec. 4 Experimentally, there have been a few studies on learning adverserially. Lowd and Meek [23] study the problem of an adversary reverse engineering classifiers, and show applications to reverse-engineering spam filters [24]. Dalvi et al. [13] present a game-theoretic analysis of how an algorithm and adversary could adapt to each other, and show applications to spam filtering. Barreno et al. [3] examine when machine learning could be more secure at a more general level, presenting a framework, and a lower bound on the work that an attacker must to evade an IDS. However, none of this work is directly applicable to our problem.

7 Conclusion

We have shown fundamental limits on the accuracy of large class of pattern-extraction algorithms in an adversarial setting. Our work generalizes earlier work on attacks which have focused on individual algorithms and current systems. We also analyzed and shown conditions under which pattern-extraction may work. Our results are applicable to other kinds of signature-generation algorithms that use easily forgeable properties of an exploit.

8 Acknowledgements

This research was supported in part by CyLab at Carnegie Mellon under grant DAAD19-02-1-0389 from the Army Research Office. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of ARO, CMU, or the U.S. Government or any of its agencies. We also thank David Brumley and Jim Newsome for useful discussions and comments on earlier versions of this paper.

References

[1] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[2] P. Auer. Learning nested differences in the presence of malicious noise. *Theoretical Computer Science*, 185(1):159–175, 1997.

[3] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. Can machine learning be secure? In *Proceedings of the ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2006.

[4] D. Brumley, J. Newsome, D. Song, H. Wang, and S. Jha. Towards automatic generation of vulnerability-based signatures. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2006.

[5] N. H. Bshouty, N. Eiron, and E. Kushilevitz. PAC learning with nasty noise. In *Algorithmic Learning Theory, (ALT'99)*, 1999.

[6] S. P. Chung and A. K. Mok. Advanced allergy attacks: Does a corpus really help? In *Proceedings of 10th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2007.

[7] M. Cost, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham. Vigilante: End-to-end containment of internet worms. In 20^{th} *ACM Symposium on Operating System Principles (SOSP 2005)*, 2005.

[8] M. Costa, M. Castro, L. Zhou, L. Zhang, and M. Peinado. Bouncer: Securing software by blocking bad input. In *Proceedings of the 21st Symposium on Operating Systems Principles (SOSP '07)*, 2007.

[9] M. Costa, J. Crowcroft, M. Castro, and A. Rowstron. Can we contain internet worms? In *Proceedings of the Third Workshop on Hot Topics in Networks (HotNets-III)*, 2004.

[10] J. Crandall, Z. Su, S. F. Wu, and F. Chong. On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS)*, 2005.

[11] J. R. Crandall and F. Chong. Minos: Architectural support for software security through control data integrity. In *International Symposium on Microarchitecture*, 2004.

[12] W. Cui, M. Peinado, H. Wang, and M. Locasto. Shieldgen: Automatic data patch generation for unknown vulnerabilities. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2007.

[13] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma. Adversarial classification. In *Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2004.

[14] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee. Polymorphic blending attacks. In *Proceedings of the 15th Usenix Security Symposium (Security '06)*, 2006.

[15] M. V. Gundy, D. Balzarotti, and G. Vigna. Catch me, if you can: Evading network signatures with web-based polymorphic attacks. In *Proceedings of WOOT'07*, 2007.

[16] M. Kearns and M. Li. Learning in the presence of malicious errors. *SIAM Journal on Computing, 22(4):807-837*, 1993.

[17] H.-A. Kim and B. Karp. Autograph: toward automated, distributed worm signature detection. In *Proceedings of the 13th USENIX Security Symposium*, 2004.

[18] C. Kreibich and J. Crowcroft. Honeycomb - creating intrusion detection signatures using honeypots. In *Proceedings of the Second Workshop on Hot Topics in Networks (HotNets-II)*, 2003.

[19] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna. Polymophic worm detection using structural information of executables. In *Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2005.

[20] Z. Li, M. Shanghi, B. Chavez, Y. Chen, and M.-Y. Kao. Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resiliance. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2006.

[21] Z. Liang and R. Sekar. Fast and automated generation of attack signatures: A basis for building self-protecting servers. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS)*, 2005.

[22] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear threshold algorithm. *Machine Learning*, 2(285-318), 1988.

[23] D. Lowd and C. Meek. Adversarial learning. In *Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2005.

[24] D. Lowd and C. Meek. Good word attacks on statistical spam filters. In *Proceedings of Second Conference on Email and Anti-Spam*, 2005.

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

[26] J. Newsome, B. Karp, and D. Song. Paragraph: Thwarting signature learning by training maliciously. In *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2006.

[27] J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS)*, 2005.

[28] R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif. Misleading worm signature generators using deliberate noise injection. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2006.

[29] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proceedings of the 6th ACM/USENIX Symposium on Operating System Design and Implementation (OSDI)*, 2004.

[30] G. E. Suh, J. Lee, and S. Devadas. Secure program execution via dynamic information flow tracking. In *Proceedings of ASPLOS*, 2004.

[31] H. J. Wang, C. Guo, D. Simon, and A. Zugenmaier. Shield: Vulnerability-driven network filters for preventing known vulnerability exploits. In *Proceedings of the 2004 ACM SIGCOMM Conference*, 2004.

[32] K. Wang, J. J. Parekh, and S. J. Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2006.

[33] K. Wang and S. J. Stolfo. Anomalous payload-based worm detection and signature generation. In *Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2005.

A Proofs for Section 4

We first discuss some common elements of all proofs in this section. As described in Sec. 2.2, the adversary is in control of the malicious instances presented in the training data. The adversary's goal is for the algorithm to learn as little as possible, and make many mistakes. Thus, it is optimal for the adversary to generate all malicious instances identically, so that each instance contains all k attributes of every reflection set. Note that this does not conflict with our assumption that there is no noise in the training data, or that the adversary is required to be truthful.

Formally, in the instance space $\{I = i \in \{0,1\}^m\}$, the argument above says that adversary gives the algorithm many copies of the instance $i = \{1, 1, \ldots 1\}$ in the training pool. For example, in red-herring attacks on pattern-extraction signature generators, this instance can be thought of the initial input given to the learning algorithms: all initial instances contain all red herrings as well as the invariants. Note that the target hypothesis is selected at adversarially or at random from the set H, and all hypotheses in H are indistinguishable to the algorithm. This implies that the adversary can ensure that algorithm gains no additional information about the target bits from the training data.

A.1 Proof for Theorem 4.1

Proof. Our proof is an application of the bounds proven in [22] to our setting. For completeness, we present the whole proof here to illustrate the sequence of instances that the attacker can present, in order to force the mistakes indicated in the lower bound.

Let us assume that the algorithm can divide into bits into sets that correspond to a critical attribute and its reflections. By definition of reflection, the algorithm cannot distinguish between these k properties, even if the algorithm is powerful enough to distinguish properties into n sets of k.

We show a sequence of instances that force the algorithm to make $\log k$ mistakes, for a single reflecting set of k properties. Since there are n such sets, and no reflecting set can

information about targets in any other reflecting set, using this strategy on each of will generate $n \log k$ mistakes.

With knowledge of the deterministic algorithm, the adversary can decide where to place the target bit in the reflecting set as follows: the adversary chooses a set of t bits. If the algorithm labels it positive, it places the target bit in the other set of $k - t$ bits, otherwise it places it in this set. By observing the actions of the algorithm, the adversary has chosen a set of $\min(t, k-t)$ bits in which to place the target bit. The adversary can repeat this process until it isolates where to place the target bit.

Thus, because the algorithm is deterministic, it is equivalent to the adversary deciding which bit to choose as the target bit, rather than deciding where to place the target bit.

The adversary begins by setting $t = k/2$, i.e., it presents an instance with $k/2$ bits set. After the algorithm makes a mistake on the instance, the adversary presents an instance with $k/4$ bits from the $k/2$ bits where the target bit needs to be present. This process continues until the number of bits is reduced to 1. The ith instance presented by the adversary has $k/2^i$ bits set to 1, and forces the adversary to commit to the presence of the the target bit in one of $k/2^i$ bits, and the adversary forces a mistake on each instance. Thus, algorithm is forced to make $\log k$ mistakes on this sequence of examples.

If there are multiple critical attributes within a single reflection set, the adversary can treat this set as two separate reflection sets, each of size k, and achieve the same number of mistakes. □

A.2 Proof for Theorem 4.2

Proof. Our proof is an application of the bounds proven in [22] to our setting. For completeness, we present the whole proof here to illustrate the sequence of instances that the attacker can present, in order to force the mistakes indicated in the lower bound.

The proof is similar to Theorem 4.1. The initial instance presented by the adversary, as before, contains all attributes, or equivalently, all m bits set to 1. As before, the algorithm may be sufficiently powerful to distinguish the reflecting sets, but it cannot identify the critical attributes within each reflecting set.

The sequence of instances that the adversary presents is similar, but chosen randomly. For each reflecting set, the adversary does the following before the algorithm identifies the target bit. The adversary chooses a set of $k/2$ bits at random from all sets of $k/2$ bits, and presents an instance with this set of $k/2$ bits. Then, with probability 1/2, the target bit is present in this set. The probability that any decision given by the algorithm on this randomly chosen set of $k/2$ bits is correct is $1/2$. Thus, the algorithm has a chance of $1/2$ of making a mistake on this step.

Once the label is given by the adversary, the information about the target bit is reduced to $k/2$, as in the deterministic case. The adversary then picks a set of size $k/4$ from the set of size $k/2$ containing the target bit, This continues until the target bit is isolated, which takes $\log k$ steps. Thus, the algorithm has an expected error of $1/2 \log k$.

For every reflecting set of size k, the algorithm will make an expected $\frac{\log k}{2}$ mistakes, and so the total expected number of mistakes will be $\frac{n \log k}{2}$. □

A.3 Proof for Theorem 4.3

Proof. Our proof for the case of $t = 0$ is an application of the theorems in [1] to our problem, a restriction of the general learning problem. We extend this for $t > 0$ in our proof. For completeness, we present the whole proof here to illustrate the sequence of instances that the attacker can present, in order to force the mistakes indicated in the lower bound.

Let $t = 0$. Then, we need to show a sequence of instances such that the algorithm makes at least $n(k-1)$ mistakes. If the algorithm is allowed to make *no* mistake on non-malicious instances, it must always label an instance to be non-malicious when it is uncertain of the label of an instance.

In order to have the algorithm make a lot of mistakes, the adversary has to present a sequence of instances such that the algorithm is always forced to label it non-malicious. The adversary does this as follows: in the ith *epoch*, he picks one reflecting set C_i to focus on, and the instances presented have the bits that correspond to all the other reflecting sets (i.e., other than C_i) all set to 1 (e.g., in the first epoch, all bits that correspond to reflecting sets C_2 to C_n are always set to 1). He starts the epoch by presenting the instance with all bits in C_i set to 1, and he chooses one additional non-target bit from the current instance, and sets it to 0 to generate the instance that follows.

Thus, within an epoch, every instance received by the algorithm (subsequent to the first instance) has one fewer bit set to 1 than the previous instance. However, it does not know whether the target bit has been set to 0, by definition of the reflecting set, and therefore has to label the instance non-malicious. As the adversary does not set the target bit to 0, each instance presented to the algorithm is indeed malicious. The adversary can present $k - 1$ such instances for each reflecting set, and thus, there are $n(k-1)$ mistakes made by the algorithm.

When $t \geq 1$, the algorithm may make at most t incorrect guesses on non-malicious instances. The adversary may use the same sequence of instances as described above, and because the algorithm is deterministic, the adversary knows when the algorithm will label an instance to be malicious. The adversary can then choose the target hypothesis so that,

at that point, a non-malicious instance is presented, i.e., it is the target bit of the relevant reflecting set that is dropped at the instance (though all bits dropped earlier in the epoch are still non-target bits, as before). With this change, algorithm has then made a mistake on a non-malicious instance, but also knows the target bit for the relevant reflecting set, and will not make any more mistakes within that epoch. Thus, each mistake on a non-malicious instance in this sequence reveals the target bit of one reflecting set, but no information about any other reflecting set. When the algorithm is allowed t such mistakes, the adversary can force the algorithm to make at least $(n-t)(k-1)$ mistakes on the malicious instances. □

A.4 Proof for Theorem 4.4

Proof. Our proof for the case of $t = 0$ is an application of the theorems in [1] to our problem, a restriction of the general learning problem. We extend this for $t > 0$ in our proof. For completeness, we present the whole proof here to illustrate the sequence of instances that the attacker can present, in order to force the mistakes indicated in the lower bound.

Once again, we begin with the case of $t = 0$. Now, the adversary is allowed to make no mistakes on the malicious instances. Therefore, any time the algorithm receives an instance, it must label it malicious, unless the algorithm is certain that the instance is not malicious.

The adversary presents any non-malicious instance with n bits present, subject to the following two conditions: (1) exactly one bit is present from each reflecting set, (2) all target bits are not present in the instance (this follows by definition of a non-malicious instance). Each instance in this sequence is non-malicious, but the algorithm is forced to label it malicious: the algorithm does not have enough information to distinguish whether any particular bit present from a reflecting set is truly the target bit for the reflecting set, until a malicious instance has been presented.

More formally, let A denote the set of all instances that satisfy the above two conditions: each instance in A contains exactly n bits set to 1, and only one bit is set from each reflecting set. Let i_{mal} denote the sole malicious instance in A. The adversary presents instances $i_1, i_2, \ldots$ from $A - \{i_{mal}\}$ to the algorithm, one at a time. Define I_w to be the set of the first w instances presented, for $w < |A - \{i_{mal}\}|$. With the non-malicious instances in I_w, it is consistent for any instance in remaining in $A - I_w$ to be the malicious instance, and so the algorithm must continue to classify the next instance as malicious. Thus, the algorithm is forced to make a mistake on every instance in $A - \{i_{mal}\}$. There are $k^n - 1$ such instances, and therefore, the algorithm makes $k^n - 1$ mistakes.

A similar analysis can be applied for $0 < t < n$. We focus on $t = 1$ for simplicity. The adversary now divides the reflecting sets into two equal groups, A_1 and A_2, and each reflecting set goes into one of A_1 or A_2; so, each group A_1 and A_2 will account for $m/2$ bits. The adversary chooses instances in two phases: in the first phase, all instances set all bits from A_1 to 1, but only set one bit from each reflecting set in A_2 to 1 (so $m/2$ bits in A_1 but only $n/2$ bits in A_2). In the second phase, the roles of A_1 and A_2 are reversed: all instances set all bits from A_2 to 1, but only set one bit from each reflecting set in A_1 to 1 (so $m/2$ bits in A_2 but only $n/2$ bits in A_1). There are $k^{n/2} - 1$ non-malicious instances in each phase.

Recall that the algorithm may make at most one mistake on a malicious instance, and thus it can call an uncertain instance non-malicious at most once. Because the algorithm is deterministic, the adversary knows when the algorithm will classify an instance to be non-malicious, and chooses, ahead of time, the target hypothesis appropriately to ensure that at that point, a malicious instance can be presented.

We term the algorithm to be *non-conservative* if it labels an instance malicious in the first $k^{n/2} - 1$ instances that it sees, otherwise we term to be *conservative*. For a non-conservative algorithm, the adversary presents non-malicious instances from Phase 1 until the point where it would label an instance malicious, and then the malicious instance from Phase 1, to ensure that the algorithm makes a mistake on the malicious instance. It then presents the non-malicious instances from Phase 2 to the algorithm. With this sequence of instances, the algorithm needs to identify the $n/2$ target bits in Phase 2 without making any mistakes on the malicious instances, and every hypothesis is consistent with the instances and labels presented in Phase 1. Thus, the mistake-bound of Phase 2 reduces to the case of $t = 0$, with a target hypothesis of size $n/2$, and so a non-conservative algorithm makes at least $k^{n/2} - 1$ mistakes.

The analysis for a conservative algorithm is similar, except that the malicious instance is presented at the appropriate point in Phase 2. Thus, this algorithm has to make at least $k^{n/2} - 1$ mistakes on the non-malicious instances in Phase 1. The analysis when $t < n$ is also similar, the adversary simply divides the reflecting sets into $\frac{n}{t+1}$ groups, instead of dividing it into 2 groups, when $t = 1$. □

B Proofs for Section 5

B.1 Proof for Theorem 5.1

Proof. We prove this in two parts: we first prove that any deterministic algorithm learning a conjunction with $dn + 1$ bits may be forced to make a lot of mistakes, and then we show that there exists a sequence of non-malicious instances consistent with overlap-ratio d, so that the adversary can force algorithm to learn a conjunction with $dn + 1$ bits.

We first show that any deterministic algorithm that learns a conjunction with $dn + 1$ bits could be forced to make $(k-1)(dn+1)$ mistakes, when reflecting sets are of size k. We count only the mistakes made on malicious instances, and therefore each of instances must contain all n target bits. The adversary may generate a sequence of instances in the following manner: he starts with the malicious instance that has all bits set to 1, and in each subsequent instance, he sets one additional non-target bit (from any reflecting set) to be 0. Because the algorithm is deterministic, the adversary can choose the target hypothesis and the bit that is set to 0 at each point and ensure that the algorithm makes a mistake on each instance. This way, for each reflecting set, the algorithm will need to have $k-1$ bits set to 0 before the target bit is revealed. As this procedure can be done for each of the reflecting sets included in the learned conjunction, the algorithm makes $(k-1)(dn+1)$ mistakes.

Now, we show that all deterministic algorithms have to learn a conjunction with at least $dn + 1$ attributes. To do this, we use the following definitions. We term a *block* to be all of the bits corresponding to a reflecting set. We say that a block is set to 0 if all bits in the block are 0, and that a block is set to 1 if all bits in the block are set to 1. We will term a *zero-information* instance to be one that has d blocks set to 1, and has all the remaining $n-d$ blocks set to 0. The set K is the set of all zero-information instances.

Each instance in K may appear in normal traffic: the instance contains no more than d target bits set to 1. Each instance in K is also non-informative about the true target – all bits in the reflecting set always appear simultaneously. With this set K, if the algorithm does not have at least $dn + 1$ bits (each from a different reflecting set) in its conjunction at any point, it can be forced to make an error on a non-malicious instance: the adversary simply chooses non-malicious instance from K that satisfies the algorithm's conjunction, and forces it to make an error on a zero-information instance. This mistake reveals no additional information to the algorithm about the target hypothesis, and the adversary can force the algorithm to make it as long as the algorithm's conjunction has fewer than $dn + 1$ bits. Thus, the algorithm makes fewer mistakes if it always learns a conjunction of size at least $dn + 1$. □

B.2 Proof for Theorem 5.2

Proof. The proof is similar to that of the previous theorem, but with two modifications. In our proof, we use K, the set of zero-information instances defined in the proof of Theorem 5.1.

First, the adversary no longer knows when the conjunction used by the algorithm contains at least $dn+1$ attributes; however, he can force the algorithm to contain such a conjunction with high probability by giving the algorithm, at random points in the sequence of instances, a non-malicious instance from K. Let $\mathcal{T}$ be the event that the algorithm uses a conjunction with fewer than $dn + 1$ attributes. For any $\epsilon > 0$, if $Pr[\mathcal{T}] > \epsilon$, an instance drawn at random from K will force the algorithm to make a mistake with probability $\epsilon/\binom{n}{dn}$. Thus, there is always a constant chance of error on the non-malicious instances if $Pr[\mathcal{T}] > \epsilon$. Therefore, if the algorithm uses a conjunction with few attributes, a long sequence of random instances drawn from K could generate, in expectation, many non-informative errors on the non-malicious instances.

Now, if the algorithm tries to find a conjunction with at least $dn + 1$ attributes (and thus include attributes from at least $dn + 1$ reflecting sets), it makes at least $\frac{k-1}{k}$ mistakes in expectation for each of the $(dn+1)$ attributes. As before, the adversary starts with the malicious instance that has all bits set to 1, and in each subsequent round, picks one additional non-target bit (from any reflecting set) to set to 0 in the instance that is presented. For every reflecting set that appears in the algorithm's conjunction, the algorithm has a $1/k$ chance of making a mistake when any non-target bit is set to 0. Because the adversary can do this $k-1$ times within a single reflecting set, the expected number of mistakes is $\frac{k-1}{k}$, within one set. The adversary can ensure that $dn + 1$ reflecting sets are used with probability $1-\epsilon$, so the number of mistakes it makes, in expectation, is $(1-\epsilon)\frac{k-1}{k}(dn+1)$, for any $\epsilon > 0$. □

B.3 Proof for Theorem 5.3

Proof. Recall that C_i denotes the ith reflecting set. Let $U = \{C_i\}_i$. Without loss of generality, we will assume that the bits in the instance are reordered so that the first k bits correspond to the attributes in reflecting set C_1; the next k bits correspond to the attributes in the reflecting set C_2, and so on. Let $x_{i,j}$ be 1 if the jth property of the reflecting set C_i is present in the instance ($ik + j$th bit is 1 in the reordered instance) and 0 otherwise.

A linear separator that identifies malicious instances needs to be of the form $\sum_{i,j} w_{i,j}x_{i,j} > t$, where $w_{i,j}$ is a weight of token, and t is any fixed value with $t > 0$. For the proof, we will use K, the set of zero-information instances defined in the proof of Theorem 5.1. Let D be a set that contains exactly d-fraction of the reflecting sets. The adversary can then force the following constraints to hold at every point of time: for every D, $\sum_{a\in D}\sum_j w_{a,j} \leq t$. This is because if the constraints do not hold, the adversary can force the algorithm to make a mistake on a zero-information instance from K, and thus the algorithm makes a mistake that does not help in identifying the target hypothesis.

As in the proof of Theorem 5.1, we show how the attacker generates mistakes on the malicious instances with these constraints. The attacker constructs malicious in-

stances as follows: for each reflecting set C_i, the attacker chooses the p bits with the lowest weights, and sets the malicious instance to have these p bits to be 1.

Let q_i be the sum of the weights of the p bits for a reflecting set C_i. Then, for every set D as defined earlier, $\sum_{i \in D} q_i \leq t\frac{p}{k}$. Let $\mathcal{D}$ be the set of all such sets D. Then, we have $\sum_{D \in \mathcal{D}} \sum_{i \in D} q_i \leq |\mathcal{D}| t\frac{p}{k}$. This implies that $\binom{n-1}{dn-1} \sum_{i \in U} q_i \leq \binom{n}{dn} t\frac{p}{k}$, giving $\sum_{i \in U} q_i \leq \frac{tp}{kd}$. By setting $p \leq kd$, $\sum_{i \in U} q_i \leq t$. Thus, the attacker can send a malicious instance with the appropriate p bits set, and the algorithm will make a mistake by labelling it non-malicious.

With this mistake, the attacker has reduced the size of every reflecting set to effectively be kd from the original size of k: the algorithm now knows that the target bit has to be among the kd bits that were set in the malicious instance just presented. The adversary can recurse this procedure with the new reflecting sets, until their size has effectively reduced to 1, and this allows the attacker to force $\log_{1/d} k$ mistakes, or $\log_{1/d} \frac{m}{n}$. □

B.4 Proof for Theorem 5.4

Proof. The proof is similar to that of Theorem 5.3; however, we need to make two modifications, because the adversary does not always know the internal state of the algorithm. In our proof, we use K, the set of zero-information instances defined in the proof of Theorem 5.1.

First, the adversary no longer knows whether the constraint $\sum_{i,j} w_{i,j} \leq t/d$ is disobeyed; however, he can force it to hold with high probability by presenting the algorithm, at randomly chosen points in sequence, a non-malicious instance from K. In particular, for any $\epsilon > 0$, if $Pr[\sum_{i,j} w_{i,j} > t/d] > \epsilon$, an instance drawn at random from K will cause the algorithm to make a mistake with probability $d\epsilon$. Thus, there is always a constant chance of error on the non-malicious instances if $Pr[\sum_{i,j} w_{i,j} > t/d] > \epsilon$ – this chance of error does not approach 0 as long as $\sum_{i,j} w_{i,j} > t/d$. Therefore, if $\sum_{i,j} w_{i,j} > t/d$, an arbitrarily long sequence of random instances drawn from K could generate, in expectation, arbitrarily many non-informative errors on the non-malicious instances.

The second modification needed is that the adversary constructs a slightly different sequence of malicious instances present to the algorithm. In this situation, the adversary cannot pick the p smallest weights, since the adversary does not know the p smallest weights. Instead, the adversary picks $p/2$ weights at random, from each reflecting set, and constructs an instance with those bits set to 1, and the rest set to 0. The probability that these $np/2$ weights exceed tp/kd is at most 1/2, which means that the probability that an mistake is caused is at least 1/2, by Markov's inequality. Thus, at each step, the algorithm makes a mistake with probability 1/2, and the number of attributes in each reflecting set reduces to $kd/2$. Thus, the algorithm makes $1/2 \log_{1/2d} \frac{m}{n}$ mistakes on the malicious instances in expectation, when the constraint $\sum_{i,j} w_{i,j} > t/d$ holds. As this can be set to hold with probability $1 - \epsilon$, for any $\epsilon > 0$, the expected number of mistakes is at least $\frac{1}{2}(1 - \epsilon) \log_{1/2d} \frac{m}{n}$. □