A Close Look on n-Grams in Intrusion Detection: Anomaly Detection vs. Classification

Christian Wressnegger
idalab GmbH
Berlin, Germany

Guido Schwenk
Berlin University of Technology
Berlin, Germany

Daniel Arp
University of Göttingen
Göttingen, Germany

Konrad Rieck
University of Göttingen
Göttingen, Germany

Abstract

Detection methods based on n-gram models have been widely studied for the identification of attacks and malicious software. These methods usually build on one of two learning schemes: *anomaly detection*, where a model of normality is constructed from n-grams, or *classification*, where a discrimination between benign and malicious n-grams is learned. Although successful in many security domains, previous work falls short of explaining why a particular scheme is used and more importantly what renders one favorable over the other for a given type of data. In this paper we provide a close look on n-gram models for intrusion detection. We specifically study anomaly detection and classification using n-grams and develop criteria for data being used in one or the other scheme. Furthermore, we apply these criteria in the scope of web intrusion detection and empirically validate their effectiveness with different learning-based detection methods for client-side and service-side attacks.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*; K.6.5 [**Computing Milieux**]: Management of Computing and Information Systems—*Security and Protection - Invasive software*; I.5.1 [**Pattern Recognition**]: Models—*Statistical*

Keywords

Intrusion Detection, Machine Learning, n-Gram Models

1. INTRODUCTION

Computer security faces a constant and daily growth of new threats in the Internet. The attack vectors, types and ramifications of these threats are manifold. On the client side, malware infections pose a risk to the security of individual hosts and connected networks. Numerous types of malware are used for infecting computers at a large scale and conducting illegal activities, such as the distribution of spam messages or the theft of personal data [see 2, 12]. On the server side, a plethora of attacks target network services, which range from classic exploits against vulnerable implementations to sophisticated injection attacks, such as XSS and CSRF. These attacks are regularly used for compromising web servers and retrieving sensitive data, such as passwords and credit card numbers.

A few years ago it might have been possible to craft detection rules for these threats manually. Nowadays however, manual analysis fails to keep pace with attack development and more automation is needed to handle the overwhelming amount of novel threats. As a result, alternative lines of research have been explored for the detection of attacks, most notably methods based on machine learning. Several of these learning-based approaches build on the concept of n-grams, that is, attacks are identified by analyzing substrings of length n extracted from the observed data stream. Such n-gram models have been successfully applied for spotting malicious activity in system call traces [11, 51], packet payloads [49, 50], executable files [19, 35] and JavaScript code [24, 37].

In terms of machine learning these detection methods can be roughly categorized into one of two learning schemes: *anomaly detection* and *classification*. In the first case, a model of normality is constructed from n-grams and used to identify attacks as deviations thereof [e.g., 11, 49], whereas in the second case a discrimination between benign and malicious n-grams is learned [e.g., 35, 37]. Although both schemes have been successfully applied in a large body of previous work, little is known about when to favor one over the other in a detection task and how the distribution and characteristics of the extracted n-grams influence this choice.

In this paper we take a close look on n-gram models for intrusion detection. We study both learning schemes in detail and shed light on what makes a problem and its data suitable for one or the other setting. As part of this analysis we develop three suitability criteria, namely the *perturbation*, *density* and *variability* of n-grams, that enable us to assess whether data fits the scheme of anomaly detection or classification should be used. We study these criteria on seven common types of data used in intrusion detection, including text and binary protocols as well as system call traces and JavaScript code. In a case study on web intrusion detection, we finally validate our criteria in both learning schemes.

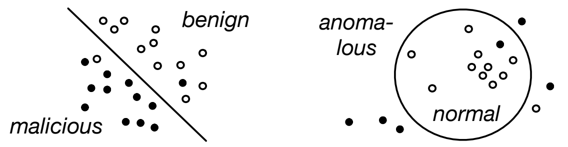
In summary, the contributions of the analysis presented in this paper are as follows:

- We discuss the scope of anomaly detection and classification for intrusion detection and define prerequisites for practical application.
- We devise suitability criteria for n-gram models for intrusion detection that help selecting an appropriate learning scheme.
- Finally, we demonstrate the validity of the developed criteria in a case study on client-side and server-side web intrusion detection.

The rest of the paper is structured as follows: The two prevalent learning schemes for intrusion detection are presented in Section 2, while n-gram models are discussed in Section 3. In Section 4 we analyze datasets from different domains and develop our suitability criteria. These criteria are evaluated in a series of experiments, whose results are presented in Section 5. Section 6 discusses related work and Section 7 concludes.

2. LEARNING SCHEMES

In many fields of application where learning methods are applicable for decision making one often is confronted with the selection of the underlying learning scheme. For intrusion detection two schemes are prevalent: *classification* and *anomaly detection*. In this section we shortly review both of these schemes in order to identify possible indicators for deciding when to use the one or the other.



(a) Classification (b) Anomaly detection

Figure 1: Schematic depiction of learning schemes.

Classification.

In computer security often very strict definitions are in demand for deciding about something being benign or malicious, which immediately suggests a *classification* task. The identification of the two classes is achieved by learning a discrimination as illustrated in Figure 1(a). Several learning methods, such as decision trees, neuronal networks and boosting can be used for learning a classification [8]. An intuitive example is the two-class SVM that learns a hyperplane separating two classes with maximum margin in a feature space [see 43]. Learning a classification, however, requires enough data of both classes in order to be able to generalize to unseen samples. If one class is represented by a few instances only, it is likely that the learning will overfit and thereby impede detection of unknown attacks. In this regard a lack of data for one class is already a crucial factor for abstaining from using classification.

In some cases of intrusion detection, sufficient data for both classes can be acquired automatically. For example, for learning a client-side detection of web-based attacks, it is possible to actively visit benign and malicious web pages using honeyclients and special crawlers [e.g., 18, 42]. This crawling enables one to assemble a recent collection of training data for both classes. In other settings, as for example the server-side detection of web-based attacks, one is restricted to passively wait for attacks using network honeypots. As a consequence, it is not possible to put together a representative set of server-side attacks in a timely manner and classification methods should not be employed.

Anomaly Detection.

Detecting unknown attacks is of critical importance in security, as these may relate to zero-day exploits or new instances of known malware. Fortunately, it is possible to take this scenario into account using *anomaly detection*—even if no attacks are available for learning. By focusing on the prominent class and learning its structure, it is possible to differentiate that class from everything else, as illustrated in Figure 1(b). Several methods are suitable for learning such a model of normality, for example, by analyzing the density, probability or boundary of the given class [8]. A common method for anomaly detection is the spherical one-class SVM (or SVDD) that determines a hypersphere enclosing the data with minimum volume [see 43].

At this point it is important to stress that anomaly detection methods do not explicitly learn to discriminate benign from malicious data, but instead normality from anomalies. This semantic gap requires one to design features and detection systems carefully, as otherwise identified anomalies may not reflect malicious activity [13, 45]. Moreover, it is also necessary to sanitize the training data to avoid incorporating attacks in the model of normality [5]. Nonetheless, anomaly detection is the learning scheme of choice if little or no data is available for the attack class, as for example, when learning a server-side detection of attacks.

Prerequisites.

In summary, both learning schemes offer their advantages if used in the right setting. We thus arrive at the following prerequisites for learning-based detection:

- **Classification.** If enough representative data is available for both classes, this scheme allows to learn a model for discriminating one class from the other. Depending on the type of attacks, this discrimination may generalize to unknown attacks but is not guaranteed to do so.
- **Anomaly Detection.** If only one class is available for learning, anomaly detection allows to learn a model for detecting unknown attacks. However, a careful design of the detection system is necessary in order to limit the semantic gap between attacks and anomalies.

3. N-GRAM MODELS

Most learning methods operate on numeric vectors rather than on raw data. Therefore, it often is necessary to construct a map to a vector space for interfacing with learning methods. In some settings, this can be achieved by defining numeric measures describing the data, such as the length or the entropy of packets. A more generic map, however, can be developed using the concept of *n-gram models*. Initially proposed for natural language processing [3, 6, 46], n-grams have become the representation of choice in many detection systems [e.g., 21, 24, 32, 37, 38, 49].

To describe data in terms of n-grams, each data object x first needs to be represented as a string of symbols from an alphabet A, where A is often defined as bytes or tokens. For example, for analyzing network packets, we simply consider the data in each packet as a string of bytes. Similarly, we can model JavaScript code in web pages by representing the code as a string of lexical tokens.

By moving a window of n symbols over each object x, we can then extract all substrings of length n. These substrings (*n-grams*) give rise to a map to a high-dimensional vector space, where each dimension is associated with the occurrences of one n-gram. Formally, this map ϕ can be constructed using the set S of all possible n-grams as,

$$\phi : x \to \big(\phi_s(x)\big)_{s \in S} \quad \text{with} \quad \phi_s(x) = \operatorname{occ}(s, x)$$

where the function $\operatorname{occ}(s, x)$ simply returns the frequency, the probability or a binary flag for the occurrences of the n-gram s in the data object x.

Several methods for the detection of attacks and malicious software indirectly make use of this map. For example, the methods PAYL [50], McPAD [32] and Anagram [49] analyze byte n-grams for detecting server-side attacks, where the first two consider frequencies and the latter binary flags for the occurrences of n-grams. Similarly, the methods Cujo [37] and PJScan [24] use token n-grams with a binary map for identifying malicious JavaScript code in web pages and PDF documents, respectively.

Although n-grams provide generic and effective means for modeling data, the exponential growth of the resulting vector space apparently impedes efficient operation. However, the number of n-grams in a particular object x is linear in the object's size and thus efficient data structures can be used for processing the extracted n-grams. For example, the method Anagram [49] builds on *Bloom Filters* [1] for storing n-grams, while the detector Cujo [37] uses sparse feature vectors to efficiently represent the n-grams. Later on we show that this sparsity is not only beneficial for dealing with high-dimensional vectors but also provides a good indicator for the feasibility of anomaly detection.

4. DATA ANALYSIS

Based on the presented n-gram models, we now study the characteristics of different types of data and their respective n-grams. A summary of the considered datasets is given in Table 1. Before we proceed to defining suitability criteria, we provide further details on the datasets and describe the different sources we gathered the data from.

4.1 Datasets

In order to cover a large variety of data from different fields of applications we consider two scenarios: First, data for *client-side intrusion detection* and second, data for *server-side intrusion detection*. The first datasets cover system call traces of programs and JavaScript code of web pages, while the latter involve binary and text-based network protocols.

Client-side Datasets.

The JavaScript dataset is gathered by randomly drawing 230,000 URLs from the list of the top most visited web pages provided by Alexa[1]. For analysis, the URLs are actively crawled using the client-side honeypot ADSandbox [7] in order to (a) extract all JavaScript code and (b) analyze the code dynamically. The output of this analysis—events monitored during the execution of the code—constitutes the first JavaScript dataset *(JS-Dyn)*. For the second dataset *(JS-Stat)* we additionally process the plain JavaScript code similarly to Rieck at al. [37] by extracting lexical tokens, such as identifiers and numerical constants. We scan all URLs using the *GoogleSafeBrowsing* service to limit the number of attacks in the data.

The dataset of system call traces *(Syscalls)* is extracted from the original DARPA IDS evaluation datasets [27] and corresponds to audit trails of the Solaris BSM module. Although outdated and criticized [28, 30] we still consider this dataset useful for studying the characteristics of n-gram models on client-side data.

Server-side Datasets.

The application layer of the Internet protocol suite contains a variety of different protocols based on which we can build our study. In particular we consider DNS and SMB as representatives for binary protocols and HTTP and FTP for text-based protocols.

The DNS dataset has been recorded in a period of one week at our research institute resulting in a total of 294,431 requests. In case of SMB an overall amount of 22.6 GB of data in 154,460 messages has been collected on a single day at our institute. SMB often encapsulates files and consequently involves far more data than other protocols. The HTTP data has been recorded from a content management system running at our institute in a period of one week and incorporates 237,080 HTTP requests. For the FTP communication we make use of data recorded at the Lawrence Berkeley National Laboratory (LBNL) during a period of 10 days with more than 22,000 FTP sessions [31].

On top of the distinction between client-side and server-side detection we differentiate the datasets according to the used type of n-grams. For our analyses and experiments we use the types as shown in Table 1, which are consistent with previous work [see 16, 37, 38, 49]. Specifically, we use byte n-grams for the network protocols and token n-grams for the JavaScript and system call datasets, where the tokens correspond to events, lexical tokens and system call identifiers, respectively.

4.2 Analysis and Discussion

When it comes to the application of learning-based methods there are several prerequisites to be clarified beforehand. For instance, how much training data is needed in order to learn an expressive model? Obviously the more data is used for training, the better the outcome might be. To detail this rather general statement Wang et al. for instance specify the "*likelihood of seeing new n-grams as training time increases*" [49] in order to indicate the completeness of their model in terms of the amount of training data. If this measure converges, the training set can be considered as sufficiently large.

If one looks at an even earlier stage of learning other very important considerations need to be made. For instance, is the underlying problem, the data to be used and the constructed feature space suitable for learning? Unfortunately previous work often falls short of explaining the reasons for or against particular methods and features.

[1]Alexa Top Sites, `http://www.alexa.com/topsites`

Dataset	*HTTP*	*FTP*	*JS-Stat* & *JS-Dyn*	*Syscalls*	*DNS*	*SMB*
Size	237,080 reqs	22,615 sessions	230,000 URLs	25,797 traces	294,431 reqs	154,460 msg blocks
***n*-gram type**	byte	byte	token	token	byte	byte

Table 1: Description of the datasets used for our analyses and experiments. Additionally to the size of the datasets also the used n-gram types are specified.

4.2.1 Suitability Criteria

We have worked out three criteria for a problem's suitability for being learned in an anomaly detection setting: The *perturbation* within a dataset's main class, the *density* of the used feature space and the *variability* of the n-grams in the individual classes. In this section we study these criteria exemplarily on the following datasets: *FTP*, *HTTP*, both views on the JavaScript code (*JS-Dyn* & *JS-Stat*) and the system call traces (*Syscalls*).

Criteria 1 (Perturbation). *The perturbation is the expected ratio of n-grams in a benign object that are not part of the training data.*

A perturbation value of 0 means that all possible n-grams in the dataset have been observed in the training phase, whereas a high value indicates that despite training a large number of unseen n-grams is still to be expected in benign objects during testing.

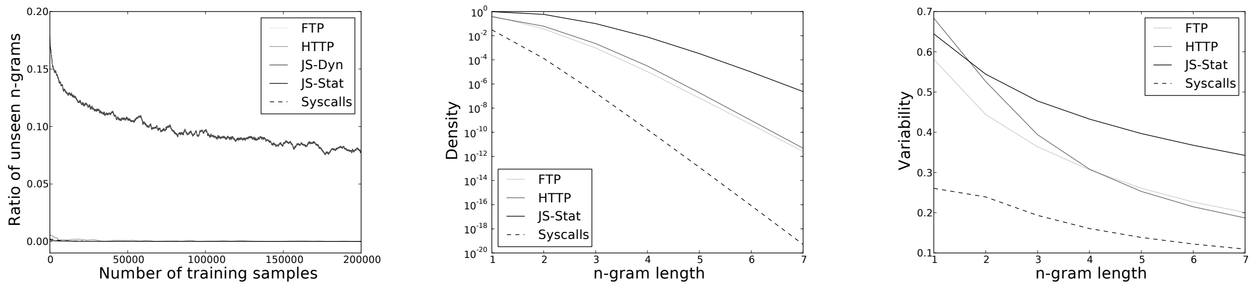
This measure is closely related to the likelihood of unseen n-grams as discussed by Wang et al. [49]. However, convergence of the likelihood only tells one that the training does not improve anymore with an increasing amount of data. It does not allow to draw conclusions about the data's suitability for anomaly detection as such. To do so we consider the likelihood of the last objects observed during the training phase as the expected perturbation during testing. Figure 2(a) illustrates the perturbation for our five datasets on the example of 3-gram models. Note that one of those clearly stands out, namely the datasets composed out of the dynamic JavaScript reports. The other four quickly converge to zero and do not exhibit any significant level of perturbation after training. Also *JS-Dyn* seems to converge but to a value unequal to zero. Hence, in this particular case benign data constantly exhibits 5–10% of unseen n-grams. This renders anomaly detection very difficult, as each benign object appears to be anomalous to 5–10% already.

So, where do the perturbations in the *JS-Dyn* dataset come from? The dataset covers the behavior of JavaScript code and thus contains variable names and strings. As a result, the alphabet of the n-grams is not fixed, that is, there exists an infinite number of tokens, consisting of variable names and strings. This is of course not a flaw of the sandbox, but rather a feature as the behavior is monitored with great detail. The resulting reports simply are not suitable for being learned on with token n-grams in an anomaly detection setting. For classification in turn Rieck et al. [37] show that the dataset can be used with great success.

However, in order to be able to still use anomaly detection one needs to preprocess the data such that the parts causing the perturbation are abstracted. For *JS-Stat* this was done by lexically analyzing the raw program code and introducing dedicated string tokens that hide but describe the raw data [24, 37]. That way also the names of functions, parameters and variables are abstracted. Similarly it is possible to abstract reports from *JS-Dyn*.

Criteria 2 (Density). *The density is the ratio of the number of unique n-grams in a dataset to the total number of all possible n-grams induced by the underlying alphabet.*

As second criteria we use the *density* of a training dataset when mapped into the feature space induced by n-grams. A value close to 0 indicates low density, i.e. the feature space is sparse, whereas a value of 1 means that the feature space is maximally dense. This is directly related to the overall size of the feature space and can provide some indication of how well a class can be learned for anomaly detection. For instance, if the datasets have been oversimplified in the course of lowering the perturbation, it might happen that the remaining symbols are too general to reflect the characteristics that differentiate one class from the other. Therefore, in feature space both classes occupy an identical and above all, dense region. This of course assumes a certain homogeneity of the benign and malicious data.



(a) Perturbation (b) Density (c) Variability

Figure 2: The suitability criteria on example of the *FTP*, *HTTP*, *JS-Dyn*, *JS-Stat* and the *Syscalls* datasets. (a) The perturbation within a dataset (averaged on a sliding window of 5000 samples), (b) the density of the introduced feature space and (c) the variability of the n-grams in these datasets.

Figure 2(b) shows the density of the *HTTP*, *FTP*, *JS-Stat* and *Syscalls* datasets. Obviously, it is not possible to measure the density for an infinite large set of n-grams as induced by the dynamic JavaScript reports. Therefore, the *JS-Dyn* dataset is not present in the figure. Using the density it is possible to estimate how well it can be learned on a particular dataset. In Section 5 we experimentally show this relation based on the *HTTP* and *JS-Stat* datasets. At first sight the plotted values seem vanishingly low due to logarithmic scale and the overall size of the feature space as denominator. However, the steepness of the density's decay over increasing n-gram lengths is the crucial indicator here.

Criteria 3 (Variability). *The variability is the ratio of the data's entropy to its maximal value as induced by the alphabet size (normalized entropy).*

The third suitability criteria is the *variability* of a dataset to be used for training. A value of close to 0 means that the variability of the data is very low, whereas a value of 1 indicates maximal uncertainty about the next element in a sequence. The corresponding values for our datasets are illustrated in Figure 2(c). As the variability is normalized to its maximal value, consequently it again cannot be determined for data that has a virtually infinite set of unique n-gram tokens such as the dynamic JavaScript reports (*JS-Dyn*). Therefore, this dataset is yet again spared out in the figure for this criteria.

Using the variability it is possible to characterize the structure of the datasets. It is equally important for anomaly detection as well as classification that the data possesses noticeable structure which can be learned and used for detection. Random data which by nature does not present any kind of visible structure, but largest possible variability would appear as a flat line at the maximum of 1.0. For the other datasets (from *JS-Stat* over *HTTP* and *FTP* down to *Syscalls*) a constant decay of the variability levels can be observed. This directly mirrors the difficulty to learn a model based on these datasets.

4.2.2 Case Study: Binary-based Protocols

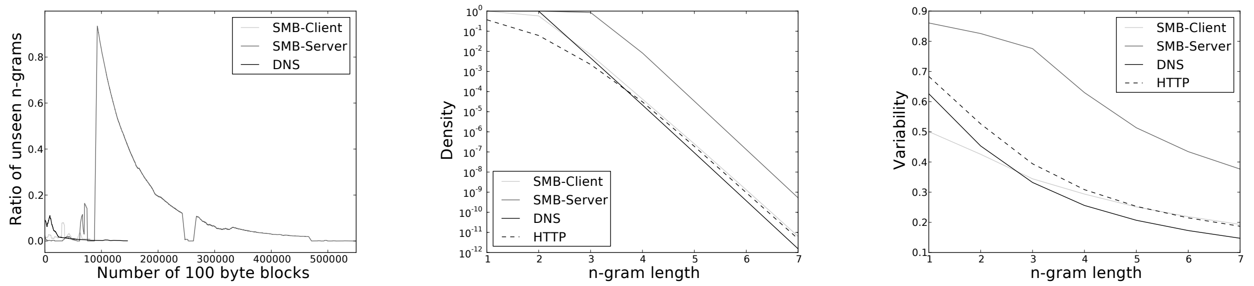
In this section we examine binary-based protocols with respect to the criteria set up in the previous section in more detail. Such protocols are not to be categorically ruled out only because they happen not to be human readable and therefore, less intuitively comprehensible. Depending on the specific dataset it might be necessary to parse and generalize the protocol beforehand, though. However, for this study and in line with previous work we apply byte n-grams on the raw payloads, similarly to HTTP and FTP.

In particular we examine the SMB/CIFS protocol suite and the DNS protocol. SMB is mainly used for the transfer of data, but includes other functionality such as the Windows RPC (*MSRPC*). Our SMB dataset features the special property that data was mainly retrieved from the server and only in a few exceptions uploaded by the client. This allows us to study two different scenarios in the use of SMB: First, the use of largely pure SMB requests send by the client to the server and only little additional raw data (*SMB-Client*) and second, SMB commands that are heavily interleaved with the transmitted data (*SMB-Server*). DNS on the other hand consists out of relatively short requests and in a large part of printable characters—the requested domain name.

In both cases the basis for a low perturbation is provided due to the bounded alphabet of 256 bytes. Figure 3(a) illustrates the perturbation levels for *DNS* and the SMB datasets. Especially *DNS* quickly reaches zero perturbation, whereas *SMB-Server* requires much more data and also *SMB-Client* would have needed more training. Note that *SMB-Server* not only stands out clearly, but its peak is displaced with respect to the beginning of the recording. This indicates that in the beginning similarly to *SMB-Client* mainly pure message blocks are exchanged, before at some point the transmission of large amounts of raw data starts.

Such raw data often is compressed and therefore exhibits a high entropy. Therefore, the uncertainty about the overall observed n-grams increases and obscures the structure of the protocol. Figure 3(c) shows the variability and reveals the lack of perceptible structure of the *SMB-Server* dataset. *SMB-Client* on the other hand does not fully share this problem. This suggests that learning n-gram models of SMB traffic would be feasible if the interleaved raw data, which constitutes additional noise for the learning algorithm, is excluded. This is equally true for classification as well as anomaly detection, whereby it is especially critical for the latter. One option to restrict the data to a manageable subset is, for instance, to only look at Windows RPC messages [15].

The density exposes another interesting property of the considered binary protocols. Figure 3(b) shows that for n-grams of lengths up to $n = 3$ the density is especially high.



(a) Perturbation (b) Density (c) Variability

Figure 3: The suitability criteria applied to the binary-based protocols in our datasets: (a) Perturbation, (b) Density and (c) Variability. For the latter two we additionally included the HTTP communication dataset as a baseline.

This happens due to the use of numeric variables as part of the protocol that may span over the entire range of byte values, such as length specifiers for subsequent fields, headers, etc. With larger values of n this influence decreases. *SMB-Client* and the DNS requests even align with the values for the *HTTP* dataset.

In summary, the developed criteria largely confirm previously expressed concerns regarding the suitability of binary protocols and especially SMB [e.g., 15] for learning. However, simultaneously it is shown that learning cannot be ruled out categorically. Constraining the use of raw data in SMB can lower the complexity to a feasible level. Also DNS requests appear to be very well manageable. Nevertheless, one needs to note that in the case of DNS the type of attacks usually differ from those seen for HTTP and FTP. For DNS it is often about the sequence and chronology of requests in order to implement denial-of-service or spoofing attacks. Therefore, although the DNS requests' data as such is suitable for classification as well as anomaly detection, learning such attacks can only succeed if timing information and relations of the requests are included in the data.

4.2.3 Pitfalls

Finally we want to point out a particular pitfall that generally comes with the use of hash functions and is related to the density criteria described earlier. For intrusion detection based on n-grams this is of special interest whenever methods make use of hashing to represent n-grams [e.g 49].

When using hash functions it is necessary to take the influence of collisions into account. Two or more objects (e.g. n-grams) may by chance result in the same hash value. Depending on the size of the feature space this is more or less likely, but of course also the overall number of hashed and store objects influences the number of collisions. This can be thought as the *saturation* of the hash function's output range. In the worst case an n-gram model may describe all possible n-grams (when fully saturated) rather than those representing normal or malicious behavior, thereby artificially increasing the density of the feature space (cf. Section 4.2.1).

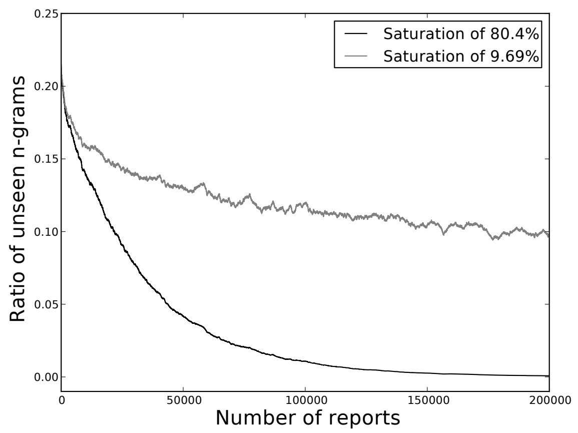


Figure 4: The influence of the saturation of a hash function's output range on the example of *5*-grams extracted from the dynamic JavaScript reports (*JS-Dyn*).

Figure 4 shows the influence of the saturation of a hash function's output range on the convergence of the dataset's perturbation. One curve shows a seemingly ideal convergence for *JS-Dyn* (*5*-grams), whereas the other suggests a far from optimal behavior. On closer examination it becomes clear that this happens due to different hash saturation levels of 80.44% and 9.69% respectively. Hence, we have made sure not to exceed a hash saturation of 10% for our analyses in order to minimize collisions and obtain the most accurate results.

5. EXPERIMENTS

We proceed with an empirical evaluation of the proposed suitability criteria. In particular, we study the performance of n-gram models for web intrusion detection. To this end, we focus on three of the datasets from Section 4.1, namely the *HTTP* dataset, comprising requests to a web server, and the *JS-Stat* and *JS-Dyn* datasets, covering analysis reports of JavaScript code retrieved by a web client.

5.1 Malicious Datasets

For the HTTP dataset, we select several recent exploits contained in the Metasploit framework [29] as attacks. These exploits are used with different payload encoders, resulting in a total of 89 HTTP attack samples. We run and record these attacks under controlled conditions against the same content management system from which we have collected the benign HTTP data. For JavaScript datasets, we use 609 attacks collected using the Wepawet service [4]. These attacks cover drive-by downloads of different types and campaigns, such as malicious web pages involved in SQL injection attacks and spam campaigns. The attacks are described in more detail in [37].

5.2 Detection Methods

We consider two anomaly detection and classification methods for our experiments: First, we employ a one-class and a two-class version of the detection method *Anagram*, and second, we make use of a one-class and two-class *SVM*s. Both methods have been successfully applied for network intrusion detection [e.g., 32, 37, 49].

Anagram Detector.

Originally designed for analyzing packet payloads only, Anagram [49] has turned into a generic anomaly detection method that found its way into different approaches and domains [e.g., 17, 20]. Anagram extends the method PAYL [50] and uses a Bloom filter for analyzing high-order n-grams of bytes. During the training phase the methods stores each observed n-gram in the Bloom filter, while for testing it computes the fraction of previously unseen n-grams in an object using lookups in that filter. Anagram can be extended to also support classification by using a second Bloom filter trained on malicious content. For our experiments we thus train one filter for benign data and another one for known attacks. For anomaly detection we only consider the benign filter of Anagram, whereas for classification we combine the scores of both filters to obtain a joint decision.

Support Vector Machines (SVMs).

As second learning method we consider SVMs which can be used for anomaly detection as well as classification [41].

Dataset	**1C-Anagram**			**2C-Anagram**			**1C-SVM**			**2C-SVM**			
	1%	0.1%	0.01 %	1%	0.1%	0.01 %	1%	0.1%	0.01 %	1%	0.1%	0.01 %	**FP**
HTTP	100%	93.0%	93.0%	95.7%	93.9%	83.8%	100%	100%	98.2%	100%	100%	100%	
JS-Dyn	37.9%	5.7%	0.0%	95.4%	81.6%	81.5%	41.6%	0.1%	0.0%	98.1%	71.2%	71.2%	
JS-Stat	94.8%	87.1%	63.1%	96.2%	83.2%	82.9%	50.9%	23.1%	14.8%	99.3%	97.4%	94.6%	

Table 2: The detection performance (as true positives rates) of the four different detection methods on our datasets. In particular we choose a *4*-gram model for *HTTP* as well as *JS-Stat* and use *3*-grams for *JS-Dyn*.

In particular, we consider the regular two-class SVM that learns a separating hyperplane with maximum margin and the corresponding one-class SVM [40]. Note that this one-class SVM is equivalent to the spherical one-class SVM discussed earlier if used with an appropriate kernel function. The decision function of both schemes then is given by

$$f(x) = \sum_{s \in S} \phi_s(x) \cdot w_s + b$$

where ϕ is the map as define in Section 3 and w the weight vector representing the hyperplane with bias b. To enable an efficient learning with high-dimensional vectors, we apply feature hashing [44], that is, the vector space is indexed using the hash values of each n-gram. This acceleration technique resembles the mechanism of Bloom filters used in the Anagram detector.

5.3 Experimental Setup

For all experiments we randomly split the benign and malicious datasets into two partitions in a ratio of three to one, considering them as *known* and *unknown* data. The latter is strictly used for performance evaluation, whereas the other one is used for training. The splitting is repeated 10 times and the results are averaged. For the malicious datasets we additionally take care that the partitions contain distinct attack types. In case of HTTP this can be done according to the 28 exploits generated by Metasploit, while for the JavaScript attacks we utilize the labels of a popular anti-virus scanner, resulting in 30 distinct attack types.

As described in the previous section Anagram is defined over the binary embedding of n-grams. For the SVM experiments we similarly to Anagram make use of that embedding. As a matter of fact our experiments show that in line with previous work [e.g., 37, 49] this embedding often performs better than a map based on n-gram frequencies. Additionally, in order to avoid a bias on the object size we normalize the feature vectors $\phi(x)$ to one, i.e., $\|\phi(x)\|_2 = 1$.

5.4 Results and Discussion

The results of our experiments are listed in Table 2. For each method and setting the true-positive rate is given for different false-positive rates: 1%, 0.1% and 0.01%. The anomaly detection scheme is prefixed with **1C** and the classification scheme with **2C**, respectively.

We can note at first, that for each dataset at least one combination of n-grams and a learning scheme performs well, such that over 81.5% of the attacks can be identified with 0.01% false alarms.

JavaScript (JS-Dyn).

Using our criteria from Section 4.2 we identify this dataset of dynamic JavaScript reports as the least suitable for anomaly detection. The conducted experiments substantiate this finding and our criteria. The difference between classification and anomaly detection is obvious (shaded in dark gray in Table 2) for both tested detection methods. The 2-class implementation of Anagram as well as the SVM clearly outperform their anomaly detection counterparts by 50–60 percent points and more. This result confirms our analysis and conclusions about this dataset and simultaneously backs up findings from previous work [37], where the effectiveness of classification in this setting was shown.

JavaScript (JS-Stat).

For this dataset of statically analyzed JavaScript code our criteria show that it is very well possible to use anomaly detection, but learning might prove difficult. This is indicated by the higher density of the dataset and slower decay thereof over increasing length of the n-grams. Also the variability is higher than for datasets such as HTTP, for which anomaly detection provably works very well [see 38, 49]. In our results this is especially notable for anomaly detection using SVMs for which we experience a similarly significant drop as for the dynamic reports. The Anagram implementation is not influenced to the same extend. However, at a second glance the detection performance for anomaly detection with Angram (1C-Anagram) drops measurably faster in comparison to classification (2C-Anagram) when it comes to smaller thresholds for the false-positive rate. If we further increase the n-gram length this effect intensifies. For instance, for 7-grams in the anomaly detection setting the true-positive rates drop to 47.8% and 15.9% for false-positive rates of 0.1% and 0.01%, respectively.

Web Requests (HTTP).

It does not come as surprise that both, anomaly detection and classification work well for the *HTTP* dataset. As summarized in Table 2 this holds true for the Anagram detector as well as the detection using SVMs. The ability to successfully detect anomalies in web requests has been proven true in several research works before [e.g., 33, 38, 49, 50] and is in line with our criteria.

5.4.1 Summary

In our experiments we show that it is not only a black-and-white decision about whether one or the other learning strategy can be used for a particular dataset, but also about the used parameters and methods. If a specific setting out of learning method and parameters works satisfactory (c.f. Anagram on *JS-Stat* using 4-grams) there is no point to push forward in a region where it does worse (7-grams). However, detailed analysis of the datasets can—next to the fundamental suitability for a specific learning scheme—also provide an insight about which parameter range might be usable.

6. RELATED WORK

The use of n-grams is popular throughout various disciplines and fields. In the following we provide an overview of n-gram models in computer security and highlight approaches using anomaly detection and classification. Moreover, we discuss work that addresses properties and limits of n-grams in detection systems.

Anomaly Detection using n-Grams.

One of the first methods for the detection of attacks using n-grams has been devised by Forrest et al. [11, 16]. The method detects anomalous program behavior by counting the number of unknown n-grams in system call traces. Following this seminal work, several related approaches have been proposed for host-based anomaly detection, for example, using probabilistic models [51], rule-based learning [25], one-class SVMs [9], neural networks [14] and the inclusion of system call arguments [22].

On the network side, first approaches for anomaly detection using n-grams have focused on byte frequencies, that is, 1-grams over bytes [23, 50]. An example is the method PAYL [50] that embeds packet payloads in a vector space and detects anomalies by computing the distance to the average 1-gram distribution. Several methods extend this work to support high-order n-grams [e.g., 17, 32, 38, 49]. For example, the method Anagram [49] makes use of Bloom filters for efficiently determining the fraction of unseen n-grams in packet payloads, while the method by Rieck and Laskov [38] builds on Trie data structures for computing distances in the resulting vector space. Similarly, the system McPAD [32] uses an approximation of high-order n-grams for efficiently learning an ensemble of one-class SVMs.

A further example for anomaly detection using n-grams is the method PJScan [24], which analyzes n-grams of lexical tokens from JavaScript code. In contrast to other work, PJScan learns a one-class SVM on malicious instances of JavaScript code and thereby realizes a reverse anomaly detection, where everything deviating from the learned model is identified as benign.

Classification using n-Grams.

A large body of security research has studied classification using n-grams. For example, Kolter and Maloof [21] evaluate several classification methods, such as decision trees, boosting methods and SVMs for detecting malicious executable files. To reduce the dimensionality of the induced vector space, the authors restrict the analysis to 500 n-grams selected using the information gain. Similarly, Reddy and Pujari [36] attempt to extract "relevant n-grams" for learning a classification between benign and malicious files. As a further extension, the method McBoost [34] precedes the classification of malware with an unpacking stage that allows to extract n-grams from obfuscated code. In a similar line of research, Jacob et al. [19] make use of n-grams over bytes to classify malware without unpacking it.

Inspired by the frequent use of word n-grams in information retrieval, several authors have also explored the use of n-grams over larger alphabets for detecting malicious data. The method Cujo [37], for instance, uses n-grams of lexical tokens for classifying benign and malicious JavaScript code with an SVM, while the method Malheur [39] conducts a classification using n-grams of events observed during the monitoring of malware in a sandbox.

Analysis of n-Gram Models.

With the increasing use of n-grams in detection systems, a branch of security research has started to explore evasion against this representation. As a result of this work, Wagner and Soto [48] introduce mimicry attacks evading anomaly detection based on system call sequences and thereby subvert n-gram models. The blending attacks by Fogla et al. [10] share the same idea but operate on the network level and are capable of thwarting the method PAYL.

Apart from these adversarial settings, however, there is little work on the characteristics of n-gram models and the selection of a learning scheme. As one example, Lee and Xiang [26] make use of information-theoretic measures to quantify what types of probabilistic methods are applicable for anomaly detection. In the course of that they identify a relation between the conditional entropy and the length of n-grams. The influence of the n-gram length is further studied by Tan and Maxion [47]. In a thorough analysis they identify blind spots in anomaly detection systems and reason about the optimal length of n-grams.

More recently Hadžiosmanović et al. [15] compare the performance of multiple anomaly detection systems, including Anagram and McPad, on a number of binary protocols. Although insightful in terms of the performance of the individual systems, the authors unfortunately missed out on explaining the reasons for the observations made.

7. CONCLUSIONS

The detection of novel threats is a challenging and enduring problem of computer security. While methods based on n-grams cannot generally eliminate this problem, they provide means for automatizing the generation of detection models and thereby help to better fend off novel attacks. However, several of the proposed solutions are rather ad hoc and there is little reasoning about why a particular n-gram model or learning scheme is used.

This paper aims at improving this situation and provides insights on n-gram models for intrusion detection. To this end, we have studied the use of anomaly detection and classification techniques with n-grams. As result of our analysis, we define prerequisites that allow to decide whether one of the two schemes is applicable (Section 2). Moreover, we develop three suitability criteria that can be computed from n-gram data prior to the design of a detection method (Section 4). These criteria enable a practitioner to assess the complexity of the detection task and help to select an appropriate learning scheme.

Our suitability criteria, however, only provide indications for favoring one scheme over the other and should not be considered alone for designing a detection method. Depending on a particular detection task, it may be possible to operate a learning scheme in a slightly imperfect setting for the sake of other constraints, such as run-time performance. Nonetheless, the criteria can guide the development of detection methods and help to avoid tedious experiments with different learning schemes.

To support further research in this field, the implementations and the prototype for conducting our analysis are open-source software and will be made publicly available at `http://mlsec.org/salad`.

Acknowledgments

The authors would like to thank Marco Cova for making the JavaScript attack dataset available. Furthermore, the authors gratefully acknowledge funding from the German Federal Ministry of Education and Research (BMBF) under the project PROSEC (FKZ 01BY1145).

References

[1] Bloom, B. Space/time trade-offs in hash coding with allowable errors. *Communication of the ACM 13*, 7 (1970), 422–426.

[2] Caballero, J., Grier, C., Kreibich, C., and Paxson, V. Measuring pay-per-install: The commoditization of malware distribution. In *Proc. of USENIX Security Symposium* (2011).

[3] Cavnar, W., and Trenkle, J. N-gram-based text categorization. In *Proc. of SDAIR* (Las Vegas, NV, USA., Apr. 1994), pp. 161–175.

[4] Cova, M., Kruegel, C., and Vigna, G. Detection and analysis of drive-by-download attacks and malicious JavaScript code. In *Proc. of the International World Wide Web Conference (WWW)* (2010), pp. 281–290.

[5] Cretu, G., Stavrou, A., Locasto, M., Stolfo, S., and Keromytis, A. Casting out demons: Sanitizing training data for anomaly sensors. In *Proc. of IEEE Symposium on Security and Privacy* (2008), pp. 81–95.

[6] Damashek, M. Gauging similarity with *n*-grams: Language-independent categorization of text. *Science 267*, 5199 (1995), 843–848.

[7] Dewald, A., Holz, T., and Freiling, F. ADSandbox: Sandboxing JavaScript to fight malicious websites. In *Proc. of ACM Symposium on Applied Computing (SAC)* (2010), pp. 1859–1864.

[8] Duda, R., P.E.Hart, and D.G.Stork. *Pattern classification*, second ed. John Wiley & Sons, 2001.

[9] Eskin, E., Arnold, A., Prerau, M., Portnoy, L., and Stolfo, S. *Applications of Data Mining in Computer Security*. Kluwer, 2002, ch. A geometric framework for unsupervised anomaly detection: detecting intrusions in unlabeled data.

[10] Fogla, P., Sharif, M., Perdisci, R., Kolesnikov, O., and Lee, W. Polymorphic blending attacks. In *Proc. of USENIX Security Symposium* (2006), pp. 241–256.

[11] Forrest, S., Hofmeyr, S., Somayaji, A., and Longstaff, T. A sense of self for unix processes. In *Proc. of IEEE Symposium on Security and Privacy* (Oakland, CA, USA, 1996), pp. 120–128.

[12] Franklin, J., Paxson, V., Perrig, A., and Savage, S. An Inquiry Into the Nature and Causes of the Wealth of Internet Miscreants. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*.

[13] Gates, C., and Taylor, C. Challenging the anomaly detection paradigm: A provocative discussion. In *Proc. of New Security Paradigms Workshop (NSPW)* (2006), pp. 21–29.

[14] Ghosh, A., Schwartzbard, A., and Schatz, M. Learning program behavior profiles for intrusion detection. In *Proc. of USENIX Workshop on Intrusion Detection and Network Monitoring* (Santa Clara, CA, USA, Apr. 1999), pp. 51–62.

[15] Hadžiosmanović, D., Simionato, L., Bolzoni, D., Zambon, E., and Etalle, S. N-gram against the machine: On the feasibility of the n-gram network analysis for binary protocols. In *Recent Adances in Intrusion Detection (RAID)* (2012), pp. 354–373.

[16] Hofmeyr, S., Forrest, S., and Somayaji, A. Intrusion detection using sequences of system calls. 151–180.

[17] Ingham, K. L., and Inoue, H. Comparing anomaly detection techniques for HTTP. In *Recent Adances in Intrusion Detection (RAID)* (2007), pp. 42 – 62.

[18] Invernizzi, L., Benvenuti, S., Comparetti, P. M., Cova, M., Kruegel, C., and Vigna, G. EvilSeed: A guided approach to finding malicious web pages. In *Proc. of IEEE Symposium on Security and Privacy* (2012).

[19] Jacob, G., Comparetti, P. M., Neugschwandtner, M., Kruegel, C., and Vigna, G. A static, packer-agnostic filter to detect similar malware samples. In *Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)* (2012), pp. 102–122.

[20] Jang, J., Agrawal, A., , and Brumley, D. ReDeBug: finding unpatched code clones in entire os distributions. In *Proc. of IEEE Symposium on Security and Privacy* (2012).

[21] Kolter, J., and Maloof, M. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research (JMLR)* (2006).

[22] Kruegel, C., Mutz, D., Valeur, F., and Vigna, G. On the detection of anomalous system call arguments. In *Proc. of European Symposium on Research in Computer Security (ESORICS)* (2003), pp. 326–343.

[23] Kruegel, C., Toth, T., and Kirda, E. Service specific anomaly detection for network intrusion detection. In *Proc. of ACM Symposium on Applied Computing (SAC)* (2002), pp. 201–208.

[24] Laskov, P., and Šrndić, N. Static detection of malicious JavaScript-bearing PDF documents. In *Proc. of Annual Computer Security Applications Conference (ACSAC)* (2011), pp. 373–382.

[25] Lee, W., Stolfo, S., and Chan, P. Learning patterns from unix process execution traces for intrusion detection. In *Proc. of AAAI Workshop on Fraud Detection and Risk Management* (Providence, RI, USA, 1997), pp. 50–56.

[26] Lee, W., and Xiang, D. Information-theoretic measures for anomaly detection. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy* (2001), Proc. of IEEE Symposium on Security and Privacy, pp. 130–143.

[27] Lippmann, R., Cunningham, R., Fried, D., Kendall, K., Webster, S., and Zissman, M. Results of the DARPA 1998 offline intrusion detection evaluation. In *Recent Adances in Intrusion Detection (RAID)* (1999).

[28] Mahoney, M., and Chan, P. An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection. In *Recent Adances in Intrusion Detection (RAID)* (2004), pp. 220–237.

[29] Maynor, K., Mookhey, K., Cervini, J., F., R., and Beaver, K. *Metasploit Toolkit*. Syngress, 2007.

[30] McHugh, J. Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. 262–294.

[31] Pang, R., and Paxson, V. A high-level programming environment for packet trace anonymization and transformation. In *Proc. of Conference on Applications, Technologies, Architectures and Protocols for Computer Communications* (2003), pp. 339–351.

[32] Perdisci, R., Ariu, D., Fogla, P., Giacinto, G., and Lee, W. McPAD: A multiple classifier system for accurate payload-based anomaly detection. *Computer Networks 5*, 6 (2009), 864–881.

[33] Perdisci, R., Gu, G., and Lee, W. Using an ensemble of one-class SVM classifiers to harden payload-based anomaly detection systems. In *Proc. of International Conference on Data Mining (ICDM)* (2006), pp. 488–498.

[34] Perdisci, R., Lanzi, A., and Lee, W. Classification of packed executables for accurate computer virus detection. *Pattern Recognition Letters 29*, 14 (2008), 1941–1946.

[35] Perdisci, R., Lanzi, A., and Lee, W. McBoost: Boosting scalability in malware collection and analysis using statistical classification of executables. In *Proc. of Annual Computer Security Applications Conference (ACSAC)* (2008), pp. 301–310.

[36] Reddy, D. K. S., and Pujari, A. K. *N*-gram analysis for computer virus detection. 231–239.

[37] Rieck, K., Krueger, T., and Dewald, A. Cujo: Efficient detection and prevention of drive-by-download attacks. In *Proc. of Annual Computer Security Applications Conference (ACSAC)* (Dec. 2010), pp. 31–39.

[38] Rieck, K., and Laskov, P. Detecting unknown network attacks using language models. In *Proc. of International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)* (July 2006), pp. 74–90.

[39] Rieck, K., Trinius, P., Willems, C., and Holz, T. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security (JCS) 19*, 4 (June 2011), 639–668.

[40] Schölkopf, B., Platt, J., Shawe-Taylor, J., Smola, A., and Williamson, R. Estimating the support of a high-dimensional distribution. *Neural Computation 13*, 7 (2001), 1443–1471.

[41] Schölkopf, B., and Smola, A. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[42] Schwenk, G., Bikadorov, A., Krueger, T., and Rieck, K. Autonomous learning for detection of javascript attacks: Vision or reality? In *Proc. of ACM CCS Workshop on Artificial Intelligence and Security (AISEC)* (Oct. 2012), pp. 93–104.

[43] Shawe-Taylor, J., and Cristianini, N. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[44] Shi, Q., Petterson, J., Dror, G., Langford, J., Smola, A., and Vishwanathan, S. Hash kernels for structured data. *Journal of Machine Learning Research (JMLR) 10*, Nov (2009), 2615–2637.

[45] Sommer, R., and Paxson, V. Outside the closed world: On using machine learning for network intrusion detection. In *Proc. of IEEE Symposium on Security and Privacy* (2010), pp. 305–316.

[46] Suen, C. N-gram statistics for natural language understanding and text processing. *IEEE Trans. Pattern Analysis and Machine Intelligence 1*, 2 (Apr. 1979), 164–172.

[47] Tan, K., and Maxion, R. "Why 6?" Defining the operational limits of stide, an anomaly-based intrusion detector. In *Proc. of IEEE Symposium on Security and Privacy* (2002), pp. 188–201.

[48] Wagner, D., and Soto, P. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*.

[49] Wang, K., Parekh, J., and Stolfo, S. Anagram: A content anomaly detector resistant to mimicry attack. In *Recent Adances in Intrusion Detection (RAID)* (2006), pp. 226–248.

[50] Wang, K., and Stolfo, S. Anomalous payload-based network intrusion detection. In *Recent Adances in Intrusion Detection (RAID)* (2004), pp. 203–222.

[51] Warrender, C., Forrest, S., and Pearlmutter, B. Detecting intrusions using system calls: alternative data methods. In *Proc. of IEEE Symposium on Security and Privacy* (1999), pp. 133–145.