The Ecosystem of Detection and Blocklisting of Domain Generation

LEIGH METCALF and JONATHAN M. SPRING, Carnegie Mellon University

Malware authors use domain generation algorithms to establish more reliable communication methods that can avoid reactive defender blocklisting techniques. Network defense has sought to supplement blocklists with methods for detecting machine-generated domains. We present a repeatable evaluation and comparison of the available open source detection methods. We designed our evaluation with multiple interrelated aspects, to improve both interpretability and realism. In addition to evaluating detection methods, we assess the impact of the domain generation ecosystem on prior results about the nature of blocklists and how they are maintained. The results of the evaluation of open source detection methods finds all methods are inadequate for practical use. The results of the blocklist impact study finds that generated domains decrease the overlap among blocklists; however, while the effect is large in relative terms, the baseline is so small that the core conclusions of the prior work are sustained. Namely, that blocklist construction is very targeted, context-specific, and as a result blocklists do no overlap much. We recommend that Domain Generation Algorithm detection should also be similarly narrowly targeted to specific algorithms and specific malware families, rather than attempting to create general-purpose detection for machine-generated domains.

CCS Concepts: • Security and privacy → Intrusion/anomaly detection and malware mitigation; Distributed systems security; • Networks → Application layer protocols;

Additional Key Words and Phrases: DNS, DGA, evaluation, metrics, blocklists

ACM Reference format:
Leigh Metcalf and Jonathan M. Spring. 2021. The Ecosystem of Detection and Blocklisting of Domain Generation. Digit. Threat.: Res. Pract. 2, 3, Article 19 (June 2021), 22 pages.
https://doi.org/10.1145/3423951

1 INTRODUCTION
In the context of cybersecurity, adversaries use Domain Generation Algorithms (DGAs) to create many Domain Name System (DNS) names. The purpose of using algorithmically generated names is, broadly, so the adversaries can circumvent defensive measures such as blocklisting. The first widespread successful use of a
DGA for this purpose was Conficker in 2008 [27]. Since then, malware authors have implemented a variety of different DGA. Cybersecurity defenders have, in turn, attempted to detect this pattern of malicious behavior. We broadly categorize these defensive detection attempts as DGA Detection Algorithms (DGADAs).

Many academic authors seem to take DGA as merely indicative of maliciousness. We take a more interpretable view and define domain generation as a behavior that may or may not be against security policy. Specifically, we define a DGA as the algorithmic generation of valid effective Second-level Domain (eSLD) DNS names. Examples of effective SLDs are example.co.uk or example.com. We define an effective Top-level Domain (TLD) as any zone on the Public Suffix List [24], and an eSLD is the zone immediately below such an effective TLD. Restricting the definition of DGA to effective second-level domains prevents conflating the issue with benign generation of internal names for network management by an organization, content distribution network, or cloud service provider—these are not DGA as we are discussing it.

An organization may decide that visiting algorithmically generated domains is against its security policy. We call our view more interpretable, because it maintains a clear definition of what is malicious. Much prior work on DGA does not define maliciousness at all. Such definitions are the purpose of organizational security policies, so the decision whether or not to adopt is largely based on an organization’s risk appetite and technical feasibility of enforcement. Risk appetite is up to the organization, but from a risk perspective, we recommend adopting a security policy that blocks DGA, because we are unaware of benign uses of DGA as we define it.

This article will provide information about the technical feasibility of enforcement. One way of enforcing this security policy would be to make an exhaustive list of generated names, and block them. However, there are about $10^{100}$ possible domain names (see Section 3.4). Therefore, blocklisting all the possible generated names explicitly is not viable. A more viable approach is to have some more condensed decision logic that can classify names as DGA or not without having to exhaustively list the names a priori. We term any such decision logic a DGADA. DGADAs may be simple rules or uninterpretable deep-learning systems.

Thus, the research question is how closely one can estimate a human decision (that is, creation via generation) from domain orthography. This attempt is, in some sense, a type error: the registrant’s creation decisions are not defined by orthographic properties, only correlated with them based on human conventions. Some DGADAs use properties of the name’s usage and registration dynamics to determine whether it is part of a DGA. There is good reason to do this during incident analysis or other retrospective analysis of names. However, DNS resolvers often work on millisecond resolution timelines. The network delay from looking up other properties of the name is, in such environments, impractical. Therefore, it is of practical importance to understand the performance and limitations of DGADAs that use only the orthography of the name itself. For this reason, this article will only evaluate DGADAs or features thereof that exclusively use properties of the orthography of the name.

This article makes two contributions. First, we will analyze the effectiveness of 19 DGADAs pairwise against distinct lists of DGA names from 30 malware families and other sources with known provenance. Second, we will assess how blocklisting DGA names impacts the conclusion that the contents of different blocklists essentially do not overlap or intersect [18, 20, 22, 23].

What we will not do is also important. We are not using blocklists to evaluate DGADA effectiveness, or vice versa. A DGADA is a kind of blocklist, even though it is represented as decision logic over the space of domain names rather than as an explicit list. Blocklists do not overlap, because there is an efficient resale market for criminal infrastructure [30] and adversaries use different infrastructure for different purposes. It is economical for adversaries to make this service distinction, because some types of activity (for example, sending spam) have a high probability of quick detection whereas some types of activity (for example, command and control) benefit from more stable or long-lived network identifiers. This conclusion is supported by studying the same blocklist across different such adversary activities—even within the same blocklist there is limited overlap among the names or IPs identified via different adversary activities [33]. When we study the intersection between DGADAs and other blocklists, we expect that they will not overlap, because a DGADA is just another kind of blocklist, and blocklists are observed to not overlap. Therefore, it would be an analytic error to use low overlap to say...
that either the DGADA or blocklist was underperforming. Rather, low overlap would strengthen the claim that blocklists identify a particular kind or kinds of defined behavior from a specific vantage and that adversaries use different infrastructure to conduct different behaviors in different locations.

Ethics considerations. Security research, if it discusses potential harms, almost exclusively tries to fit into a human-subjects research framework for approval. We have no human subjects, but there may still be reasonable concerns about containment or harm from the research. The main concern is whether our results benefit adversaries more than defenders. We are evaluating open source tools without attribution to any users of those tools with open source data, so we are not providing any targeting information to adversaries. We have not done anything that a moderately competent adversary could not do, since all of our sources are public. The benefit to the defender understanding of the state of play is more important.

The article is organized as follows. Section 2 discusses prior work on both DGA and blocklist ecosystem analysis, in Sections 2.1 and 2.2, respectively. Section 3 covers the effectiveness of available DGADAs. Section 3.4 presents the study methodology and Section 3.5 presents the results. Section 4 analyzes the (lack of) overlap among DGADAs and how it relates to the lack of overlap among blocklists generally. Section 5 discusses the impact and interpretation of these results. Section 6 lays out future work, and Section 7 summarizes our conclusions.

2 RELEVANT LITERATURE

Adversaries generate domain names for a variety of purposes, and a common defensive reaction is to create and sometimes share blocklists of such malicious, suspicious, or unwanted domains. The context of our DGA work is bounded by other uses and detection of obtaining domains in high volume, such as typosquatting. The blocklist ecosystem, likewise, has important context outside domain blocklisting, such as listing IP addresses or other indicators of compromise and working on information sharing incentives. Sections 2.1 and 2.2, respectively, describe the background on these topics.

Before these specific topics, there is useful shared background on the adversaries themselves. Since the early 2010s, cybercriminals can easily be specialists in specific techniques or technologies. Criminals take part in a complex economic system in which they buy, sell, or trade skills, expertise, and services—for example, crimeware-as-a-service [30] and exploit-as-a-service [13] models are well documented. In general, adversaries appear to be constrained by their workload and are averse to work [2]. These findings are consistent with a model that the economic cost to adversaries is dominated by capital costs, as marginal costs are nearly nothing [31]. These economic facts about adversaries are important, because understanding the relationship between DGA and blocklists turns not so much on technical details but rather on what resources are limiting.

2.1 Domain Generation Algorithm

The first large-scale use of DGA was by the Conficker worm in late 2008 [27]. The Conficker.C algorithm generated 50,000 names per day, up from 250 in prior versions, to overwhelm attempts at blocking access to its command and control servers.

Initial responses to such an overwhelming volume of names to block involved reverse-engineering each DGA so a list of suspicious names could be circulated to network defenders ahead of time. However, this manual process does not scale well. There is a clear incentive for automated detection of maliciously generated names. Initial attempts used machine learning to detect the existence of DGA in traffic, for example Antonakakis et al. [3] classify and cluster DNS traffic (specifically, NXDomain errors).

We searched for a systematic review of the DGADA literature,¹ and mostly found recent machine learning approaches to malicious domain detection in general [48] and DGA detection using any methods (not just limited

¹We used Google Scholar searches for both “source:CSUR ‘domain generation algorithm’” and “‘literature review’ ‘domain generation algorithm’” on March 1, 2020.
to orthography) [45]. Our focus is more narrow than these surveys, but they do provide some useful context and references.

Zago et al. [45] provides a feature analysis of machine-learning-based DGADA. Their results inform Section 3.2. There are some shortcomings in Reference [45]. For example, they do not distinguish second-level domain from effective second-level domain, as we do using the usual method of the public suffix list [24]. They also do not specify a reference value for character frequency or ratios. Although we do not exhaustively improve the description and implementation of the features identified by Reference [45], we are able to take some of the more promising ones and better contextualize and ground them.

Some studies have documented what malware authors do. Plohmann et al. [26] and Bambenek [7] are two good examples of such studies. Plohmann et al. [26] taxonomized 43 DGA based on the generation seed and scheme. The more easily detectable DGA use deterministic, arithmetic generation methods. This may influence the fact that 36 of the 43 DGA Plohmann et al. [26] catalog fall into these categories. However, Bambenek [7] is a free operational list described as “listing of all known DGA generated domains used by malware for domains 2 days prior to 3 days after the current data.” It does not categorize the generation method, but does tag the malware family. Both provide lists of names observed to be generated by malware DGA, but Bambenek [7] is continuously updated; we will use it as a data source.

Vinayakumar et al. [38] presents a comparison study of deep-learning-based DGADA that use only features of domain orthography. Among the seven deep-learning approaches tested, Woodbridge et al. [41] often but not always outperformed the others as measured by the area under the ROC curve as well as $F_1$ score.

Woodbridge et al. [41] also uses Bambenek [7] and Alexa to evaluate their DGADA. The tests in Vinayakumar et al. [38] are not directly comparable to our evaluations. However, since Woodbridge et al. [41] presents measurements against Bambenek [7] and Alexa, there are some cases where their tests are comparable to our results. Since Woodbridge et al. [41] is one of the best deep-learning methods, as measured by Vinayakumar et al. [38], we use it as a comparable best available deep-learning approach. Woodbridge et al. [41] notices some of the differences in DGA method among malware families that we report, but they do not analyze it in the context of multiple DGADAs nor the context of other blocklists.

Defenders have built further automatic DGA detectors, but automated reconstruction of the exact generation algorithm has not been possible in general. Therefore, the core problem identified by Antonakakis et al. [3, p. 13] largely remains:

“once a new DGA is discovered, [the detector] can build fairly accurate statistical models of how the domains generated by the DGA ‘look like,’ but it is unable to learn or reconstruct the exact domain generation algorithm. Therefore, [the detector] will generate a certain number of false positives and false negatives.”

There are two natural questions, given a variety of DGADAs. First, the extent to which the various detectors agree on what constitutes a DGA. Second, how well the DGADAs perform against various lists of names, both benign and malware-generated. These are our two DGA-related research questions; Section 2.2 will introduce a further research question on the impact of what we learn on the blocklist ecosystem.

2.2 Blacklist Ecosystem

The blocklist ecosystem consists of hundreds of lists of indicators of compromise. The main indicators are DNS names, IP addresses, and hashes of files. Because this article focuses on DGA, we will summarize what is known of the domain-name blocklist ecosystem; however, the results are essentially the same for IP addresses and hashes.

The headline take-away is that blocklists do not, for all intents and purposes, overlap. This is despite the fact that there are dozens of lists containing between tens of thousands and tens of millions of unique names.

---

2We thank an anonymous reviewer for this reference, as it is more recent than the available systematic literature reviews.
More specifically, between 95% and 98% of names on blocklists are on exactly one blocklist over any given 6-month period. This result has been robust to various potential confounding factors—for example, there is still not substantial overlap after using passive DNS to map blocklisted names to the IP addresses hosting them. Our understanding of the blocklist ecosystem is informed by a series of empirical studies over six years, including Metcalf and Spring [22], Metcalf et al. [23], and similar work by Kührer et al. [18], Verizon [37], and Li et al. [20].

There are several explanations that help account for this surprising result. The economic factors discussed above, such as specialization in cybercrime service markets, produce systematically different uses of names for different purposes. These distinct sets of names for different purposes are then detected differently by blocklist creators based on variation in sensor vantage [8] and detection strategy. A detection strategy is what the list administrator is looking for, such as outbound Zeus command and control domains, inbound SSH attacks, or specific features of abusive email. These differences are further exacerbated by the fact that abuse reporting is largely volunteer and reactive [16]. Finally, the blocklist creators are incentivized to be risk averse. It is better to be precise and avoid false positives that might break their customer’s system by blocking a key common domain name or system file.3

The question we aim to address in this article is whether this non-overlap among blocklists is actually due to DGA. For example, if the different blocklists actually just see different results from the same DGA, the lists are actually tracking the same latent variable, the DGA. If this were the case, then the non-overlap is merely expected statistical noise. However, if general DGA detection is not possible or different blocklists track different specific DGA, then the economic and incentive-based explanation above still coheres.

3 DGA DETECTION ALGORITHM EFFECTIVENESS

This article studies the effectiveness of 19 DGADAs—16 prior methods, Section 3.1, and three novel ones, Section 3.2—with four binary tests; the results are compared to one additional opaque DGADA from the literature that provided the results to a comparable test. We include DGADA only if it is open source and implementable within the study environment, as Section 3.1 describes. Section 3.2 describes the three new detection algorithms we designed. Section 3.4 describes the comparison methodology. The results, described in Section 3.5, compare the different algorithms’ statistical measures of accuracy (precision and recall) in four test conditions.

The task of building a general DGA detector has been elusive. To help understand why, Figure 1 visualizes several distributions of individual English characters (unigrams) from different sample populations. The goal of DGADA is, at heart, to distinguish benign from malicious domains. However, maliciousness is not directly correlated with “randomness.” In statistical jargon, the term “random” is not well-defined. The concept may be partly captured by “drawn from a uniform distribution,” that is, all elements have an equal probability of being selected, for example Figure 1(b). Shannon [29] demonstrated that characters in English, and other natural languages, are not drawn from a uniform distribution. However, this does not mean domains with distributions of characters compatible with the uniform distribution are malicious. And domains with characters drawn from a statistically similar distribution as English are not necessarily benign. And of course, domains that are similar to neither the English nor uniform distributions, such as Chinese, may also be either benign or malicious. With this background, we move next to our DGA evaluation method.

3.1 Algorithm Curation Method

The first step in our method is to describe the search criteria for discovering and selecting DGADAs. We then implemented the selected methods in our environment. The second part of this section will describe our test data sets. We also conducted what is essentially a second complementary study, on the interaction between DGA with common and blocklisted domain names, which we describe in Section 4.

---

3This error has happened with a few anti-virus engines, for example Panda in 2015 [40].
We found DGADAs via web search and via construction. The web search interval was for any time prior to August 31, 2017; the search was performed on September 1, 2017. We used google.com with the following search terms and combined the results: DGADetection, DGADA, and DGA defense. We test any results available by that date against our exclusion criteria.

We applied the following pragmatic exclusion criteria to reduce the resulting candidate to an actionable set. The algorithm had to be open source. The goal of this article was not to implement all abstractly defined DGADAs. If the algorithm was open source, then we attempted to use it in our study. Two open source algorithms were removed, because they had some sort of fatal error during installation that could not be easily traced (Wylie [43] and Zhang et al. [47]). We discarded one open source method, because its performance was orders of magnitude slower than the other methods (Shamir [28]). In summary, the three criteria for including DGADAs in our evaluation are:

1. Was the source code available?
2. Could it run on our systems?
3. Was it self-contained?

The available algorithms that meet these criteria are:
• Bayesian bigram [35]
• Bayesian unigram [35]
• Entropy bigram [44]
• Entropy unigram [44]
• Naive-Bayesian bigram [35]
• Naive-Bayesian unigram [35]
• Probability bigram [35]
• Probability unigram [35]
• Total Variation bigram [35]
• Total Variation unigram [35]
• detector [10], DGA detection based on a trained Markov chain model and ngram analysis.
• entropy [17], threshold is 130% the mean entropy of (some set of) Alexa domains, using Shannon entropy of a word
• freq [6], character pair frequency analysis based upon static frequency tables
• ngram [5]
• r [15], a random forest model for DGA detection
• shingling [12]

These implementations are dependent on the work of one master’s thesis that implemented various DGADAs from the academic literature [34]. We have not closely scrutinized these implementations, but they appear sound. Since we will not measure performance metrics, only outcome metrics, the quality or optimization of the implementation should not effect the results of the comparison. However, this situation highlights the restrictiveness of the requirement for open source implementation. There are plenty of detection algorithms that are private. We conduct our comparison using openly available data sets, so other DGADA creators could at least repeat our measurements and more easily publish comparisons with other algorithms.

3.2 Implementation of Algorithms

The three detection algorithms we construct for this work have one common feature—they all depend on a dictionary. We limit this report to an easily parsed, open source English dictionary. Specifically, in all three methods, the dictionary we use is /usr/share/dict/words, which is Webster’s second international dictionary from 1934 with minor adjustments as documented in the man page [11]. It contains 234,936 words. Future work should expand this to other languages, and specifically punycode DNS representations of words in these languages as they might be used in internationalized domain names. There are also various other English dictionaries that might perform better or worse, see for example the Debian virtual package wordlist. We use the canonical default such English dictionary as a base line to motivate whether this approach has merit, not as a final solution.

The three algorithms have near matches to features studied by Zago et al. [45]. One difference is in our definition of DGA behavior, restricting the string tested to the effective SLD. In both www.example.com and www.example.co.uk, the eSLD is example, because co.uk functions as the effective TLD. We use the Public Suffix List to identify TLD [24]. Another difference is that we make the dictionary explicit, above, and use an open source one. Some features we implement, such as “word-like units,” Zago et al. [45] describes but no paper in their review is marked as actually using. Evaluating similarity to words is the basis for approx, below, and we believe we have done the first well-documented test of such a feature.

The DGADAs we constructed are:

String Approximation (approx). The idea behind approx is that domain names that are similar to dictionary words are not DGA, and anything else is a DGA name. This makes the assumption that the dictionary covers all the relevant languages and jargons. Our current implementation does not validate this assumption; in fact, non-English words may be tagged as DGA based on the dictionary we use. We discuss the impact of this in
Section 3.5. We perform a suggestive test of the impact of expanding the dictionary to include stable, popular domain strings in Section 3.6. A more intractable assumption is that DGA domains will not contain dictionary words. Based on the number of possible domain names, it is possible to generate names from combinations of dictionary words. In practice, while some DGA authors do this, many do not. Forcing malware authors to do it would reduce the space of names they can select from.

Defining "similar" is the hard work. Given the source dictionary, we apply standard fuzzy string matching techniques on individual eSLD labels. A label is considered to be from a DGA if it approximately matches any word in the dictionary within a matching factor of 90%.

Fuzzy string matching, also known as approximate string matching, was defined by Wu and Manber [42]. The method uses the Levenshtein edit distance [19] to approximately match strings. A match is approximate if it is within a defined closeness threshold to the target string, where closeness is defined as low Levenshtein distance. The allowed changes between strings include insertions, substitutions, and deletions; the distance is the minimal number of those actions to change one string into another. We use the Perl implementation of Fuzzy String Match [14].

The default matching factor in Hietaniemi [14] is 90%, and we expect that was chosen as a satisfactory default for many use cases. With that as a strong prior, we conducted some basic parameter space exploration; since there is only one parameter for this algorithm, it is straightforward. The results suggest 90% is optimal. For each of the pairwise tests described in Section 3.5, we tested a 20,000 element sample from each of the four data sets at 80%, 85%, 90%, and 95% matching factor. Based on this sample, the parameter did not impact the $F_1$ score at two significant digits. A factor of 90% has marginally (on the order of ±0.002) better recall and marginally worse precision than the other options, but these more or less cancel out. We maintain the default matching factor of 90%.

Vowel frequency (vowels). Most letter frequency analysis uses the entire alphabet. This method simplifies the analysis to only use vowel frequency. Following English convention, we take the character set $\{a, e, i, o, u, y\}$ to be vowels. For each word length in the dictionary, we compute a reference vowel frequency distribution. We assume a Gaussian distribution and characterize the distribution by the mean and standard deviation of vowels in words of that length, expressed as a proportion of characters in the string that are vowels.

Given an eSLD, if the proportion of characters that are vowels in the label is within two standard deviations of the mean for words of the same length then the label is considered to be human-generated. Otherwise, the method considers it to be a DGA string. By definition, this will classify about 5% of dictionary words as DGA. This method is different from the "ratio of vowels" feature identified by Zago et al. [45] in that we have a defined, open source dictionary and an explicit, interpretable threshold. We employ it as a reference measurement—any deployed DGADA will clearly need to do better than that.

Average Letter Frequency (letfreq). This method is a version of unigram frequency analysis. Given a dictionary, we calculate the frequency of each character. For example, in FreeBSD [11], A has frequency 0.0855, B is 0.0160, and C is 0.0316. Given an eSLD, we calculate the average frequency of characters it contains, using the frequencies from the reference dictionary. For example, if the eSLD is cab, then the value for the string is $\frac{0.0316 + 0.0855 + 0.0160}{3} = 0.0444$. The threshold for distinguishing machine-generated names is if the value for the string is above $\frac{1}{25}$, or 0.0385, which is by definition the average frequency given a character set with 26 characters. This method is different from the "mean of frequencies" feature identified by Zago et al. [45] in that we have a defined, open source dictionary and an explicit, interpretable threshold. Zago et al. [45, p. 14] identifies mean unigram frequency as the feature with the best information gain besides length and pronounceability, but these two concepts do not benefit from an open source reference dictionary in the same way unigram frequency does. Like vowels, this method is meant as a transparent benchmark that any DGADA should exceed.

3.3 Data Curation Method
We curated four data sets to test these DGADAs. All four data sets have known provenance; two are machine-generated names and two are human-generated names.
For the machine-generated strings, we used a public list of domains [7] generated by 33 malware families that employ a DGA, collecting all domains listed during any day in November 2017; we label this dataset as “DGA.” There are other collections of purported DGA names, such as Vinayakumar et al. [39]; the UMU DGA data certainly is an improvement on prior work as far as pre-constructed feature sets [46]. But nothing can match the operational relevance of the Bambenek [7] list that is constantly collected from actual malware.

We also generate strings between 3 and 24 characters long from a uniform distribution of the characters legal in DNS labels; per RFC 1035, these are 0-9, a-z (case insensitive), and hyphen (-) as long as it is not the first or last character. We call this generated dataset "Random." The specific algorithm for generating labels is shown in Listing 1. The maximum length of 24 matches the longest word in our English dictionary; this is much shorter than the longest legal DNS label, which is 63 characters.

**Listing 1. psuedocode for generating random eSLD labels.**

```python
initialize string str # output variable
max = int(rand(22))+3; # length between 3 and 24
for (0 to max) {
    # character set defined in RFC 1035
    list = "abcdefghijklmnopqrstuvwxyz0123456789-"
    if (starting character) {
        list = "abcdefgijklmnopqrstuvwxyz0123456789-"
    }
    if (final character) {
        list = "abcdefgijklmnopqrstuvwxyz0123456789-"
    }
    append a random char from list to str
}
return str;
```

For the human-generated strings, we used the dictionary file `/usr/dict/words` [11] and Alexa [1], which we call Words and Alexa, respectively. For the dictionary, we take unique case-insensitive words of at least three characters, since DNS is case-insensitive and the shortest string in Random is three characters. Following Metcalf et al. [21], we do not naively use a single day of Alexa’s data, there is too much churn. We gathered the Alexa top one million for the second half of 2017, July 1 to December 31, and found all the domains that were on the list for every one of those 155 days. This filtering leads to a list of the stable, popular eSLD; based on our expert analysis of a sample of these names, the filtering method produces a list that appears to be completely human-generated names. This method returned 490,855 names; we randomly sampled 200,000 of them to create the test data set.

We create or sample 200,000 strings from each of the four data sets and use each DGADA to evaluate all 800,000 strings. For the Random set, we would expect some strings to actually be dictionary words, predominantly words of three or four characters. Specifically, the chances that Random generates a word in the dictionary are as follows. Given a string length ℓ and dictionary words of that length w_ℓ, we can calculate the probability P_ℓ of a string drawn from the uniform distribution being one of the dictionary words as \( \frac{w_\ell}{36^{\ell-1}} \). For example \( P_3 = 0.03 \), \( P_5 = 0.0002 \), and \( P_{10} = 7 \cdot 10^{-12} \). In our sample, 234 generated strings were in the dictionary; we did not remove these, because it would disturb the uniform distribution. Future work might set different string length bounds. This overlap highlights the difficulty of implementing a security policy based on blocking a name based on the intent with which it was created.

### 3.4 Comparison Methodology

We constructed and tested each DGADA in Section 3.1 with each of the four data sets in Section 3.3. The DGADAs mark each domain in each data set as either DGA or not. The four tests are pairwise discrimination tests between every combination of human-generated versus machine-generated. The human-generated data sets should all be marked negative, and the machine-generated should all be marked positive. This organization allows us to calculate evaluation statistics for each pairing of human- and machine-generated data.
A general DGADA should be able to discriminate either of these machine-generated data sets from either of the human-generated data sets. A specific DGADA would only be expected to detect the algorithm used by a specific malware family. The DGADAs we test are presented as general detectors, so we test them as such. Each DGADA we are testing has a static detection model, so we do not have to account for ordering or sequencing impacting the evaluation results of a detector.

To interpret the results, we calculate precision and recall for each of the four human-machine pairings of the data sets. The four pairings are Random with Words, Random with Alexa, DGA with Words, and DGA with Alexa. Intuitively, Random and Words should be the easiest to discriminate and DGA and Alexa should be the hardest.

Precision captures the percentage of items the test identifies as machine-generated that are in fact machine-generated. Recall captures whether the test identifies every string that is in fact machine-generated, and does not miss any. Formally, $\text{pre} = \frac{TP}{TP + FP}$ and $\text{rec} = \frac{TP}{TP + FN}$, where

- $TP$ is true positives, or the number of domains from the DGA or Random set correctly flagged as machine generated;
- $FP$ is false positives, or the number of domains from the Words or Alexa set incorrectly flagged as machine generated; and
- $FN$ is false negatives, or the number of domains from the DGA or Random set incorrectly designated as human-generated.

The domain names to which we have access are not all possible or all the relevant names. To help get a sense of the variation in the response of the DGADA we bootstrap the calculation of precision and recall. Bootstrapping is an effective automatic technique for estimating the correct interval of a statistic from a biased or incomplete sample [9]. The first step is to generate 150 resamplings of 150,000 elements with replacement out of each population of 200,000, and then calculate precision and recall for each resample. We then calculate the mean, median, and standard deviation for precision and recall, for each DGADA. From this, we create a range in which the precision and recall are likely to truly reside.

These tests focus on internal validity, because we know by construction whether or not a string is generated by a DGA. We are also explicitly avoiding the question of whether machine-generated names are in fact malicious. This tactic is necessary to be able to cleanly define the test. And because the Bambenek [7] dataset is generated directly from captured malware, we have some assurance our test will be applicable to the real world. This test also remains suggestive of shortcomings in precision—if a DGADA cannot discriminate “DGA” from Words, it suggests it will not be an adequate detector of machine-generated malicious names. However, this test does not address the question of whether DGADAs are a unique kind of malicious behavior, or if they are distinct from other behaviors identified by blocklists. Section 4 focuses on this question more directly, providing a complementary measure related to the ecological validity of how DGADAs mark domains from “the wild.”

### 3.5 Results

Overall, the detection algorithms perform better when the distinction between machine-generated and human-generated strings is sharper. However, even in these artificial situations, no algorithms perform perfectly. This section presents summary results for each of the four comparative tests in turn, with a more detailed set of results for the most realistic test (DGA versus Alexa).

Figure 2 shows that nine of the DGADAs have a precision better than 0.95, but that recall is more difficult, with no detectors above 0.95 and only three above 0.9. As noted in Section 3.4, Random contains some actual words drawn by chance. Even accounting for this, the expected number of such strings is about 900 out of 200,000 or 0.0045, so there is no obvious reason why recall should be so far below 0.995, which would be the maximum given these chance-drawn real words. This means that, even in the easiest case of strings truly drawn from the uniform
distribution against strictly dictionary words, the best DGADA in our test set fails to detect more than one in 20 machine-generated domains. The best precision is 1, perfect precision, so at least in that case the DGADA is reliable—if it claims a string is machine-generated, then it is. However, the lower recall would still leave plenty of space for malicious actors to generate more names. To the extent that Random is representative of the full set of eSLD labels, recall of 0.95, for example, would mean that $0.05 \cdot 10^{97} = 5 \cdot 10^{95}$ possible labels would evade
detection. Random is not representative if the DGADA uses string length or distance, but for the other DGADAs, it is probably sufficiently representative to make these results troubling.

The approx detector has a precision of 1 by construction in this test, since the dictionary words are the same set that was used to define an approximate match. This data point is not easily visible in Figure 2(a): Because the standard deviation of the precision is 0, there is no black bounding box and the graph just shows the point (1, 0.938). Other dictionaries might contain words that do not approximately match \[1\], but the correct response in that situation is probably to update the dictionary used by approx. The circularity of this test would be more of a concern if approx performed poorly in other ways, but it has the second-highest recall in Figure 2(a) as well, after detector; and the high recall remains even when neither test element is dictionary words—0.938 in Figure 2(b) and 0.996 in Figure 2(d)—although precision suffers.

Figure 2(d) shows that the detection algorithms all perform worse when the strings are actual popular domain labels instead of dictionary words. There is no way for the detectors to know that xkcd is a popular webcomic and llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch.co.uk belongs to a village in Wales; neither are a random string. However, several of the DGADAs do worse than our naïve benchmarks. The detector vowels is centered around the point (0.848, 0.914) and letfreq around the point (0.923, 0.880). Many of the DGADAs perform worse on both measures. The cluster below 0.1 recall and 0.5 precision in Figures 2(a) and 2(b) contains four detectors: Bayesian (unigram), Naïve Bayesian (unigram), and both unigram and bigram probability analysis. Since the axes start at 0.2, they do not even appear in Figure 2(d). These four have tremendous trouble detecting that the Random set is machine-generated; this is odd as Figures 2(c) and 2(d) show they do much better at detecting that the malware DGA domains are machine generated.

Figure 2(c) shows that most DGADAs perform somewhat worse when real malware-generated strings are used as the machine-generated test set. One exception is the four detectors that performed so poorly against the Random set; they now perform nearly as well as other detectors, though still measurably worse. Another exception is approx, which is lost in the axis tick marks in the top right corner, at the point (1, 0.995). This combination of perfect precision and high recall is the best performance point of any DGADA in any of the four comparisons.

Figure 2(d) captures the most realistic of our controlled comparisons. The machine-generated case is sampled from labels actually generated by malware, and the human-generated case is sampled from the websites consistently ranked the most popular over six months. It is also the comparison on which the DGADAs tend to do worst. Only three detectors achieve above 0.8 recall: shingling, ngram, and approx. Of those, approx has the highest recall (0.996) but the lowest precision (0.581).

Since the DGA versus Alexa test is the most realistic, we provide the full numeric results from our bootstrap analyst of that comparison in Table 1. All of this information is in Figure 2(d) except for the labels associating each DGADA with its results. There is no clear consensus on whether unigram or bigram measures work better; of the five detector pairs that have both unigram and bigram analysis, bigram is clearly better in two (Bayesian and naïve Bayesian), unigram clearly better for Entropy, and the other two have mixed results. The best combination of precision and recall is ngram. None of the DGADAs have better precision than that reported by the closed-source Table 2 Woodbridge et al. [41]. Although we could not acquire that model, it is tested on Alexa domains versus DGA domains as well. Barring any systematic differences introduced by collection dates or data curation, the mean precision and recall for our tests should be comparable to the macro average data they report. It is likely we have curated Alexa differently than them, as we followed Metcalf et al. [21] and used a subset of Alexa that was stable eSLD over time and Woodbridge et al. [41] appear to simply use the top one million fully qualified domain names (for an unspecified date or date range). It is unclear which direction this would bias the precision and recall.

Overall, Table 1 presents dismaying results. Based on the comparative analysis, no open source DGADA is even close to accurate enough to be used for distinguishing malware domains from popular domains. However,
Table 1. Median, Mean, and Standard Deviation for Precision and Recall for 19 DGADAs when Distinguishing between Observed Malware DGAs and Alexa

| DGADA                   | $m_{\text{pre}}$ | $\bar{x}_{\text{pre}}$ | $s_{\text{pre}}$ | $m_{\text{rec}}$ | $\bar{x}_{\text{rec}}$ | $s_{\text{rec}}$ |
|-------------------------|------------------|-------------------------|------------------|------------------|-------------------------|------------------|
| Bayesian (bigram)       | 0.85882          | 0.85875                 | 0.00093          | 0.55135          | 0.55119                 | 0.00132          |
| Bayesian (unigram)      | 0.76093          | 0.76096                 | 0.00121          | 0.49251          | 0.49255                 | 0.00131          |
| Entropy (bigram)        | 0.84509          | 0.84532                 | 0.00152          | 0.27584          | 0.27595                 | 0.00104          |
| Entropy (unigram)       | 0.88783          | 0.88777                 | 0.00112          | 0.46394          | 0.46385                 | 0.00125          |
| Naive-Bayesian (bigram) | 0.89638          | 0.89637                 | 0.00106          | 0.49913          | 0.49922                 | 0.00133          |
| Naive-Bayesian (unigram)| 0.80705          | 0.80712                 | 0.00117          | 0.46645          | 0.46632                 | 0.00136          |
| Probability (bigram)    | 0.76933          | 0.76935                 | 0.00099          | 0.65900          | 0.65900                 | 0.00123          |
| Probability (unigram)   | 0.80663          | 0.80655                 | 0.00123          | 0.46661          | 0.46662                 | 0.00133          |
| Total Variation (bigram)| 0.77224          | 0.77224                 | 0.00116          | 0.57571          | 0.57561                 | 0.00142          |
| Total Variation (unigram)| 0.68147        | 0.68155                 | 0.00102          | 0.60616          | 0.60609                 | 0.00111          |
| approx                  | 0.58133          | 0.58136                 | 0.00042          | 0.99552          | 0.99553                 | 0.00018          |
| detector                | 0.89703          | 0.89702                 | 0.00084          | 0.76331          | 0.76332                 | 0.00108          |
| entropy                 | 0.84473          | 0.84482                 | 0.00096          | 0.68113          | 0.68115                 | 0.00116          |
| freq                    | 0.77191          | 0.77195                 | 0.00103          | 0.64242          | 0.64233                 | 0.00137          |
| letfreq                 | 0.87317          | 0.87321                 | 0.00111          | 0.47312          | 0.47310                 | 0.00130          |
| ngram                   | 0.94913          | 0.94910                 | 0.00057          | 0.87815          | 0.87809                 | 0.00082          |
| r                       | 0.91837          | 0.91842                 | 0.00102          | 0.44867          | 0.44864                 | 0.00135          |
| shingling               | 0.78640          | 0.78644                 | 0.00079          | 0.89590          | 0.89591                 | 0.00087          |
| vowels                  | 0.80525          | 0.80528                 | 0.00099          | 0.67679          | 0.67670                 | 0.00114          |
| LSTM DNN [41, Table 2]  | N/A              | 0.9674                  | N/A              | N/A              | 0.8571                  | N/A              |

We have included a 20th element, the macro average of the best deep neural network DGADA [41] identified by the relevant literature review [38].

these can all be done with the time available at name resolution. The decision times are presented in Table 2; the most expensive algorithms take about 80 ms per name. The fastest decisions come in at about 25 μs.

The four comparison tests confirm what we expected. First, malware generated domains are not strictly drawn from the uniform distribution of characters, and this makes them harder to detect. Second, popular domains are not drawn strictly from words or phrases, making them harder to ignore than dictionary words. We find no general-purpose DGADA, within our scope of open source tools at least.

Some of the DGADAs maybe be well-tailored to a specific malware’s algorithm. We expect that would work better, but it is not the claim many detectors make; more often detectors claim to be general purpose. The DGA dataset reports which malware family generates a given domain, so we can calculate the differential performance against each. Figure 3 makes it clear that certain malware families are better at evading DGADAs than others. A similar pattern holds for precision. One malware family the DGADAs struggle with the most is suppobox. Figure 1(a) likely explains this—that malware family mimics the character frequency of English closely.

One potential approach to improve the results of the open source detectors is to create an ensemble detector. Table 3 suggests that creating such an ensemble detector is not straightforward. There is no number of votes from among the DGADAs that, if we take that number as the minimum to identify a machine-generated domain, we get a satisfactory result. The best $F_1$ score is for five votes, at 0.87, closely followed by four votes at 0.86. However, the best single-algorithm $F_1$ score represented in Table 1 is $\text{ngram}$ at 0.91. It is likely possible to create a more

$^4$We calculate the score over the median scores, but otherwise it is defined as usual, $F_1 \stackrel{\text{def}}{=} 2 \cdot \frac{m_{\text{pre}} \cdot m_{\text{rec}}}{m_{\text{pre}} + m_{\text{rec}}}$.
Table 2. Classification Time Performance Data in Milliseconds (ms) for All DGADAs

| DGADA         | ms per name | DGADA         | ms per name |
|---------------|-------------|---------------|-------------|
| approx        | 68.3        | Naive-Bayesian (bigram) | 75.9        |
| Bayesian (bigram) | 76.9        | Naive-Bayesian (unigram) | 76.8        |
| Bayesian (unigram) | 76.2        | ngram         | 0.6         |
| detector      | 0.3         | Probability (bigram) | 79.2        |
| entropy       | <0.1        | Probability (unigram) | 79.4        |
| Entropy (bigram) | 82.7        | r             | 8.7         |
| Entropy (unigram) | 81.1        | shingling     | 51.0        |
| freq          | 74.4        | Total Variation (bigram) | 76.8        |
| letfreq       | <0.1        | Total Variation (unigram) | 79.1        |
|               |             | vowels        | <0.1        |

Result is the mean aggregate time to decide about a name over all 800,000 names in the test sample. The test machine is a dual-processor 6-core, 2.66 GHz with L2 Cache (per Core) of 256 KB, L3 Cache (per Processor) of 12 MB, and 64 GB of memory.

Fig. 3. Mean recall across all 19 DGADAs against each of the malware families identified by Bambenek [7] for the DGA versus words case.

nuanced ensemble. Woodbridge et al. [41], the best available neural network classifier found in the literature, is one such example; however, its precision and recall are still well short of a general detector.

Overall, one should not mistake any such ensemble detector for a general-purpose DGADA. If someone invented a new DGA, then we would expect any such ensemble technique to fail to detect it, because it is not valid to infer or expect that detecting one or more existing DGA means a method can detect other arbitrary DGA.

Any detector tailored to a specific DGA, ensemble or otherwise, is more like an efficient blocklist; the detection algorithm would occupy less disk space than a list of all the domains that the DGA could create. But such an ensemble is still a blocklist, and suffers from the fragility of blocklists in general—once the adversary knows what the defender is blocklisting, it is fairly easy to create and use a new item not on the blocklist.

The latent assumption about DGADAs seems to be that efficiently detecting millions or billions of generated
Table 3. Median, Mean, and Standard Deviation for Precision and Recall for an Ensemble of the DGADAs Based on One Vote per Algorithm with Different Voting Thresholds

| minimum votes | \(m_{\text{pre}}\) | \(\overline{x}_{\text{pre}}\) | \(s_{\text{pre}}\) | \(m_{\text{rec}}\) | \(\overline{x}_{\text{rec}}\) | \(s_{\text{rec}}\) |
|---------------|---------------------|-----------------|--------|-----------------|-----------------|--------|
| 1             | 0.52277             | 0.52278         | 0.00018| 0.99927         | 0.99927         | 7e-05  |
| 2             | 0.62592             | 0.62594         | 0.00047| 0.99287         | 0.99289         | 0.00021|
| 3             | 0.72035             | 0.72031         | 0.00069| 0.9801          | 0.98008         | 0.00038|
| 4             | 0.78464             | 0.78464         | 0.00072| 0.95927         | 0.95929         | 0.00052|
| 5             | 0.82492             | 0.82498         | 0.00068| 0.91989         | 0.91984         | 0.00074|
| 6             | 0.8472              | 0.84721         | 0.00079| 0.84689         | 0.84693         | 0.00098|
| 7             | 0.86431             | 0.86431         | 0.00077| 0.78298         | 0.78306         | 0.00109|
| 8             | 0.88382             | 0.88381         | 0.00081| 0.74703         | 0.74705         | 0.00106|
| 9             | 0.90275             | 0.90269         | 0.00084| 0.71158         | 0.71163         | 0.00109|
| 10            | 0.91786             | 0.91789         | 0.00076| 0.65947         | 0.65953         | 0.00119|
| 11            | 0.93048             | 0.93044         | 0.00079| 0.60531         | 0.60538         | 0.00118|
| 12            | 0.9419              | 0.94185         | 0.00083| 0.55298         | 0.55298         | 0.00117|
| 13            | 0.95606             | 0.95611         | 0.00069| 0.4956          | 0.49568         | 0.00139|
| 14            | 0.9727              | 0.97269         | 0.00059| 0.42763         | 0.42756         | 0.00122|
| 15            | 0.98732             | 0.98736         | 0.00046| 0.33586         | 0.33584         | 0.00116|
| 16            | 0.99555             | 0.99557         | 0.00034| 0.23544         | 0.23556         | 0.00108|
| 17            | 0.99964             | 0.99963         | 0.00012| 0.166           | 0.16613         | 0.00103|
| 18            | 1.0                 | 1.0             | 0.0    | 0.07817         | 0.07813         | 0.00075|
| 19            | 1.0                 | 1.0             | 0.0    | 0.01537         | 0.01536         | 0.0003  |

Data for distinguishing between observed malware DGAs and Alexa.

domains would be an adequate defensive solution. A clever ensemble DGADA might be able to curate and collect thousands of generation methods and detect them reliably. The problem is that there are \(10^{97}\) domains. If each detection method is fantastically successful and excludes \(10^{70}\) names, and there are 10,000 equally successful non-overlapping detection methods, then the adversaries will still have \(10^{22}\) names to readily use. For reference, global supply of IPv4 addresses has been pushed to the limit in the last decade, but there are \(4 \cdot 10^{9}\) of them.

3.6 Updates to Approximate Matching

As one example, we wanted to test how sensitive these algorithms are to configuration choices. We chose our approximate matching algorithm (approx) to test, partly because of the conceptual simplicity of the method, and partly because we wanted to try to improve its precision while maintaining its high recall. To change the definition, we updated the dictionary used by approx to include not only `/usr/share/dict/words` as described in Section 3.2 but also Alexa domains.

To build the additions to the dictionary from the Alexa list, we defined an approach consistent with a rolling dictionary. A rolling dictionary could be automatically updated, if it were useful. A stable and conservative definition of the additional strings is the eSLD of every domain that has been on the Alexa top 1 million every day from July 1, 2017 to December 30, 2017. We will hold out December 31, 2017 as a test data set. This leads to a dictionary of roughly 600,000 strings, up from 234,000.

Testing the precision of this new configuration is not directly comparable to our results in the previous section. If we tested on the Alexa data set we created, then it would already include every string. So, we take all 1 million Alexa names for December 31 as a test set for human-generated strings. This data set is considerably noisier than our curated Alexa data set used in the rest of this section, so the following results are not directly comparable.
Approximate matching using this expanded dictionary had improved precision and approximately the same recall. In the case of distinguishing the Bambenek DGA strings from the 1 million Alexa names on December 31, recall is 0.996 and precision is 0.78. Using the human dictionary, approx has 0.996 recall; the two recall values are indistinguishable within three significant digits. But against the curated Alexa set, approx has 0.58 precision.

We draw two lessons from this small test. First, machine-generated domains do not tend to be approximately similar to popular domains. Domains that are similar to popular domains would likely be typosquatting domains, of which there are many [36]. However, potential typosquats are policed relatively carefully as a brand management issue, and it appears that malware DGA do not often stumble upon those near-miss sorts of names. Despite this hopeful insight, the relatively low precision also indicates that popular domains are not particularly similar to each other, or to slightly less popular domains. The continued difficulty of false positives is a further piece of evidence against a general-purpose DGADA.

So far, we have focused on the extent to which machine-generated domains are malicious, and focused on evaluating the effectiveness of detecting whether a domain is machine-generated. Section 4 addresses the complementary question of whether malicious domains tend to be machine-generated. We will use the DGADAs studied so far to inform this question. Given the relatively weak results reviewed in this section, we will be conservative about interpreting if a given domain is malicious based on the results of any single DGADA.

4 DGA AND BLOCKLISTS

This section reports some basic statistics about the relationship between domains on common blocklists and DGA. Specifically, our goal is to understand if blocklists are mainly constructed or uniquely constructed out of generated names. This question is of interest for at least two reasons. First, if blocklist are mostly reactive collections of generated names, it would be more efficient for defenders to have DGADAs that could proactively block any name that matched the DGA, instead of listing out some subset of generated names for the blocklist. Perhaps more accurately, DGADAs are just another kind of blocklist (or expression of a blocklist) and need to be used from the appropriate vantage to complement other list sources. Second, domain generation may play a causal role in the observation that most blocklists do not contain overlapping elements [22]. This result is counter-intuitive, and one hypothesis is that if generated names were accounted for then overlap would appear.

We follow the blocklist curation methods of Metcalf and Spring [22], during the period of July 1, 2017 to December 31, 2017. Similarly, we blind the names or sources of the blocklists. We want to focus on the shape of the ecosystem, rather than support any particular list. But as far as threat type and vantage, the lists are targeted at spam-sending domains, suspicious links found in email bodies, phishing sites, malware download domains, and malware command and control. The lists are generally constructed and used for threats pertaining to North America and Europe.

From 11 blocklists, we collected a total of just under 41 million unique domain names. Table 4 reports that all except 0.04% are marked by some DGADA as machine-generated. Given the relatively low precision and recall for many detectors reported in Section 3, it is unclear how to interpret this fact. Table 5 reports the number of blocklist domains marked by each DGADA. If we combine this data with Table 1, and we assume that the malware DGA versus Alexa case is sufficiently representative of the blocklist ecosystem (a tenuous assumption), then we can calculate better estimates of how many blocklist domains are actually machine-generated.

We estimate minimum and maximum machine-generated names on blocklists as follows. Take the number of names for each DGADA in Table 5 as \( n_D \in N \). For each DGADA, an estimated minimum number of machine-generated names on blocklists is \( n_D \cdot m_{pre}^D \), where \( m_{pre}^D \) is the median precision for the DGADA as reported in Table 1. Similarly, an estimated maximum is \( n_D / m_{rec}^D \). The average minimum over \( n_D \in N \) is 20,663,059; the average maximum is 45,845,747. Since many detectors have relatively low recall, many predict total machine-generated blocklist content to be over the total size of the blocklist. If we adjust the average to exclude estimates above the total blocklist size, then the adjusted average maximum is 29,424,458. This corresponds to 50% to 72% of blocklisted names are machine-generated.

Digital Threats: Research and Practice, Vol. 2, No. 3, Article 19. Publication date: June 2021.
Table 4. Number of Blocklisted Domains That Are Marked as DGA by Exactly X Algorithms, and Percentage of Total Names Marked as DGA of 40,857,873 Names on Blocklists

| X DGADAs | blocklist names marked DGA | ratio of total |
|----------|---------------------------|---------------|
| 0        | 15,801                    | 0.0004        |
| 1        | 391,011                   | 0.010         |
| 2        | 1,939,720                 | 0.047         |
| 3        | 1,840,974                 | 0.045         |
| 4        | 1,584,919                 | 0.039         |
| 5        | 1,708,679                 | 0.042         |
| 6        | 1,264,778                 | 0.031         |
| 7        | 1,085,844                 | 0.027         |
| 8        | 1,135,647                 | 0.028         |
| 9        | 1,224,082                 | 0.030         |

While this estimate constrains the scale of the question, it does not help detect which names are machine-generated. To do that, we propose to take a majority vote of DGADAs—if at least 10 detectors classify a name as machine-generated, we consider it to be so. Based on Table 5, summing over the buckets from 10 to 19 indicates 28,666,418 domains would be considered machine-generated under this voting scheme. Based on the precision and recall for an ensemble with 10 votes in Table 3, this set of names is far from perfect. However, the size of the list is plausible based on the precision and recall estimates, and is the lowest number of votes we could require and keep the result within our adjusted maximum. A more conservative approach could use 11 or 13 votes, instead of 10. But since we are seeking to characterize the impact of DGA on the blocklist ecosystem, we take the more permissive (lower vote count) definition of DGA.

The research question is, therefore, if we remove blocklisted names that at least 10 DGADAs mark as DGA, what is the intersection relationship among the remaining non-machine-generated names on blocklists? Table 6(a) presents the intersection among the 11 blocklists without modifying the lists. Table 6(c) presents the
Table 6. Intersections Among Blocklists; Table 6(a) is the Baseline and Is Consistent with Past Work [22]

| X lists | intersection | ratio of total | X lists | intersection | ratio of total | X lists | intersection | ratio of total |
|---------|--------------|---------------|---------|--------------|---------------|---------|--------------|---------------|
| 1       | 39,921,011   | 0.977         | 1       | 28,232,624   | 0.985         | 1       | 11,688,387   | 0.960         |
| 2       | 725,387      | 0.018         | 2       | 333,139      | 0.012         | 2       | 392,248      | 0.032         |
| 3       | 133,892      | 0.003         | 3       | 66,527       | 0.002         | 3       | 65,365       | 0.005         |
| 4       | 37,629       | 0.001         | 4       | 18,995       | <0.001        | 4       | 18,634       | 0.002         |
| 5       | 15,395       | <0.001        | 5       | 7,780        | <0.001        | 5       | 7,615        | 0.001         |
| 6       | 7,161        | <0.001        | 6       | 3,711        | <0.001        | 6       | 3,450        | <0.001        |
| 7       | 2,658        | <0.001        | 7       | 1,302        | <0.001        | 7       | 1,356        | <0.001        |
| 8       | 526          | <0.001        | 8       | 275          | <0.001        | 8       | 251          | <0.001        |
| 9       | 115          | <0.001        | 9       | 49           | <0.001        | 9       | 66           | <0.001        |
| 10      | 26           | <0.001        | 10      | 16           | <0.001        | 10      | 10           | <0.001        |
| 11      | 2            | <0.001        | 11      | 0            | 0             | 11      | 2            | <0.001        |

(a) Intersection among 11 blocklists as they are published.
(b) Intersection among 11 blocklists considering only likely machine-generated domains.
(c) Intersection among 11 blocklists considering only likely human-generated domains.

intersection among them when all machine-generated names are removed from the lists first. Table 6(b) presents the complementary data; it contains the intersections among the blocklists only considering the domains labeled as a DGA by 10 or more DGADAs. Table 6 suggests that domain generation impacts the degree of overlap between blocklists. About twice as many likely human-generated names are on two or more blocklists compared to likely machine-generated domains. However, the absolute difference is rather small, since so few domains are on two or more blocklists in the first place. Even with the likely human-generated domains, 96% are unique to the blocklist that identified the domain.

Figure 3 demonstrated that DGADA performance is related to malware family. If we also suppose that blocklist construction is related to malware family, then it is likely that different DGADA correlate to certain blocklists. As a low-risk strategy, blocklist curators seem to focus on one specific domain generation source as a concrete target. This produces a reliable list of predictably malicious domains. However, blocklists generally experience heavy pressure to make low-risk predictions or measurements of maliciousness. And non-generated domains are, by the evidence in Table 6, only marginally less likely to have the same heterogeneous factors such as source and collection vantage push the blocklists to curate different content.

5 DISCUSSION

This section discusses three aspects of the results. First, how to interpret whether the descriptive statistics are adequate for operational purposes. Second, the infeasibility of generic DGADA and some potential explanations for this. Third, what the number of available DNS labels explains about the dynamics of blocklisting, DGADA, and their complementary use in network defense.

We use precision and recall as descriptive statistics; here, we will discuss how to interpret adequate values for these measures. In addition to precision and recall, we would like to see DGADA producers discuss the alarm error rates for a range of base-rates of the occurrence of suspicious or generated domains in a data set. It is not possible to understand adequate precision and recall without understanding a range for the base rate of occurrence in the intended application. The alarm error rate is the probability that an alarm (that is, a positive identification of a DGA in this case) is genuine, and is characterized as the ratio of false positives to all positives. If the base rate of occurrence of the item of interest is 0.5, then alarm error is simply 1-precision [4]. In cybersecurity, the base rate is usually more like 0.001. Most domains requested by hosts are benign, otherwise most of
the Alexa domains would be malicious. With a base rate of $\frac{1}{10^{999}}$, a test with 0.99 precision and recall will have a false alarm rate of 0.99, which is nowhere near the 0.01 one gets at a base rate of $\frac{1}{2}$. Given this need for sensitive detectors, none of the open source DGADAs are adequate detectors.

Our analysis of the DGA ecosystem indicates that, at least with open source tools, it is not possible to reliably distinguish DGA from non-generated domains. This failing is despite a decade of work. We suggest one plausible reason for this failing is that the enterprise of determining the intention of a domain name based on the arrangement of letters in the domain is a confusion of semantics and syntax. There is no formal or enforced relationship between how any individual DNS label is used and the string that signifies that label. Humans give these strings natural language meanings often in the general case; example.com has a syntax that is related to its function as an example. But adversaries easily can (and do) subvert these norms. One common use case for detecting generated domains is to block fast flux networks. We suggest that if the defensive goal is to detect fast flux, the defender needs to detect the structural features of fast flux directly; there are open source tools for this purpose [21]. A structural approach matches how a resource is used with its allowed or intended use, which is clearly linked to an expressible security policy.

Our framework to evaluate the detection methods includes how many strings, or more formally DNS labels, the DGADA would prevent an adversary from using. A useful detector need not block all possible generated or malicious strings, but it should complement reactive detection methods such as blocklisting. This goal is more difficult than it may appear due to the number of available labels.

There are about a googol ($10^{100}$) possible labels. According to RFC 1035, a legal label contains 63 ASCII characters or fewer, where the legal characters are 0–9, a–z, and mid-label hyphens (–). For example, let us calculate the possible eSLD labels in one TLD, such as .com. There are 26 possible first characters (a–z), 36 possible last characters (a–z and 0–9), and 37 possible interior characters (alphanumerics plus hyphen, “-”) in a label. Thus, we can calculate all the possible 5-character labels by looking at how many options one has for each character in the label. For a 5-character label, there are $26 \cdot 37 \cdot 37 \cdot 36$, or $26 \cdot 36 \cdot 37^3 = 48,728,160$ labels. For longer labels, continue multiplying by 37 for each additional character. The total possible labels is the sum of those for each length between 1 and 63. In notation, the equation becomes

$$26 + \sum_{n=2}^{n=63} (26 \cdot 36 \cdot 37^{n-2}) = 4.400 \cdot 10^{98}.$$  

$10^{98}$ is a staggering number. It is a quadrillion times the number of atoms estimated to exist in the universe. Each eSLD is available in each of 1000 TLD, so we could perhaps justify saying there are a googol ($10^{100}$) of eSLD. One might contend that 63-character labels are too large to be worth considering in full. The longest label in the Bambenek data is 44 characters, and the longest word in our English dictionary is 24 characters. Even so, a 22-character label has about $2^{114}$ options, which is enough in 2020 to be a cryptographically secure key length (according to NIST SP 800-131Ar2). If we drastically cut the length to 11 characters, then there are still $10^{17}$, or $2^{57}$, possible labels. This tiny fraction of the name space is still so big as to be nearly incomprehensible; if each label is about 10 bytes on average, at 10 ASCII characters, then a list of all possible 11-character labels would take up just under two exabytes (a million terabytes). DNS blocklisting via Request Policy Zones is efficient, but a production name server is unlikely to have even a terabyte of memory for storing and using blocklists. The supply of potential domain names is fantastically large.

This huge supply creates an economic imbalance. If the supply of a commodity is one googol, then no wonder the cost is nearly free, as malicious domains are observed to be [31]. The price falls as the supply increases. Based on this, it is no wonder blocklisting is not a complete solution. A computer can blocklist at best a trillion names explicitly, or 0.0001% of the possible 11-character names. But the adversary can always just move to another name. A googol less a few billion names is still near enough to one googol as to make no matter. This failure of
blocklisting by itself is one main motivation for DGADAs, so one might better judge the detection algorithms based on how much of this oversupply of labels they can efficiently exclude.

6 FUTURE WORK
We identify at least two areas for future work. The first is investigating the interaction between adversarial machine learning and DGADAs. The second is compensating for the limitations of our study in regard to data sources and data collection duration; more broadly, this involves improvements to generalizability by making more observations.

It is unclear the extent to which machine learning used to classify DGA is subject to misclassification attacks that are common against other topic areas, such as image or audio classification [25]. Due to this and other factors, best practices for designing and deploying machine learning systems that will interact with adversaries or their tools, such as DGA, are not well established, though there has been some progress [32]. Operationalizing any DGADA that is based on machine learning and resistant to adversarial manipulation is an open area for future work.

Our study has limitations. We have stronger internal validity than external validity. We have good confidence that our results are free from confounds, especially in the case of the dictionary words and strings generated from the uniform distribution. We trust the labels on our data. There is a clear trade off with generalizability or external validity of our results; our decision to limit the generated string length, for example, makes the generated strings systematically not representative of all possible domain names. As Section 5 discussed, there are $10^{97}$ possible labels, so taking a representative sample from that population will always be difficult. Likewise, the Bambenek DGA list and the Alexa popular domain lists are what they are, and within the experiment we have treated them appropriately. But, almost certainly, neither is properly representative of the whole population of machine or human-generated domains, respectively. Similarly, the second half of 2017 is surely not properly representative of other time periods. The theme of these limitations is in our labeled data.

We believe that our research method could be reused as-is, if better-labeled data is available. Our selection of statistical techniques, namely, bootstrapping to acquire ranges for our precision and recall estimates, is designed to account for and be tolerant of these uncertainties or inconsistencies in data sources. However, longer longitudinal samples and further types of domain names would be valuable future work. If DGADAs are a kind of blocklist, and (as the blocklist literature shows) blocklists do not overlap due to economic pressures on adversaries (see Section 2.2), then we should expect a broader longitudinal study to be consistent with our findings.

7 CONCLUSION
We have demonstrated that open source Detection Algorithms are not adequate for fully generally detecting machine-generated domains. We also demonstrate that blocklists contain a lot of machine-generated domains, but that this does not alter the conclusion of prior work that blocklists are largely distinct and track different non-overlapping phenomena. From this perspective, a specific DGADA can profitably be considered just another blocklist in the blocklist ecosystem. One plausible explanation to unify these observations is that the available space of domain names is so large that different generation algorithms tend to produce non-overlapping results, and so any adequate detection method must be targeted to a specific DGA. Creating a general DGADA is extremely unlikely to work, but creating a series of DGADAs targeted at specific malware families is tractable. Unfortunately, creating detectors in response to adversary actions puts the defender in the same economic situation as with adding individual domains to blocklists—the adversary will continue to profit [31]. Detection and

---

5There are two areas of research that may be called “adversarial machine learning.” We are using the term to identify research about how adversaries can attack ML systems and defenses therefrom, not training ML models via a competitive interaction.
blocking of the structural behavior defenders want to prevent, such as fast-flux networks [21], is not a panacea but is an overlapping defense that should be employed in addition to detecting specific DGA.

ACKNOWLEDGMENTS

We are grateful to several NASK folks, including Piotr, for pointing out the interaction between DGA domains and blocklists as a research area. The authors thank Eric Hatleback and Allen Householder for their assistance, in particular, with their contributions of the methods “Average Letter Frequency” and “Vowel Frequency,” respectively.

REFERENCES

[1] Alexa. 2013. Alexa Internet, Inc.—Top sites. Retrieved from http://www.alexa.com/topsites.
[2] L. Allodi, F. Massacci, and J. M. Williams. 2017. The work-averse cyber attacker model: Theory and evidence from two million attack signatures. In Proceedings of the Workshop on the Economics of Information Security.
[3] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon. 2012. From throw-away traffic to bots: Detecting the rise of dga-based malware. In Proceedings of the USENIX Security Symposium. USENIX, 491–506. Retrieved from https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/antonakakis.
[4] S. Axelsson. 2000. The base-rate fallacy and the difficulty of intrusion detection. ACM Trans. Info. Syst. Secur. 3, 3 (2000), 186–205.
[5] M. Baggett. 2018. freq.py. (2018). Retrieved from https://github.com/sans-blue-team/freq.py.
[6] M. Baggett. 2016. freq.py. SANS. Retrieved from https://github.com/sans-blue-team/freq.py.
[7] J. Bambenek. 2019. Domain feed of known DGA domains from −2 to +3 days. Retrieved from https://osint.bambenekconsulting.com/feeds/dga-feed.txt.
[8] M. Collins. 2015. Using vantage to manage complex sensor networks. In Proceedings of the 11th Annual FloCon. Software Engineering Institute, Carnegie Mellon University.
[9] T. J. DiCiccio and B. Efron. 1996. Bootstrap confidence intervals. Stat. Sci. (Aug. 1996), 189–212.
[10] exp0se. 2015. dga_detector. Retrieved from https://github.com/exp0se/dga_detector.
[11] FreeBSD. 1993. README file for /usr/share/dict/words. Retrieved from http://web.mit.edu/freebsd/head/share/dict/.
[12] D. S. Gokul Das. 2013. Dga-detector. Retrieved from https://github.com/dsusin/dga-detector.
[13] C. Grier, L. Ballard, J. Caballero, et al. 2012. Manufacturing compromise: The emergence of exploit-as-a-service. In Proceedings of the Conference on Computer and Communications Security. ACM, 821–832.
[14] J. Hietaniemi. 2013. String::Approx—perl extension for approximate matching (fuzzy matching). Retrieved from https://metacpan.org/pod/String::Approx.
[15] J. Jacobs. 2016. DGA. Retrieved from https://github.com/jayjacobs/dga.
[16] M. H. Jhaiveri, O. Cetin, C. Gañán, T. Moore, and M. V. Eeten. 2017. Abuse reporting and the fight against cybercrime. ACM Comput. Surveys 49, 4 (2017), 68.
[17] L. Kirkpatrick. 2017. DGA detection. Retrieved from https://community.rsa.com/community/products/netwitness/blog/2017/02/01/dga-detection?comment=33497.
[18] M. Kührer, C. Rossow, and T. Holz. 2014. Paint it Black: Evaluating the Effectiveness of Malware Blacklists. Technical report TR-HGI-2014-002, Ruhr-Universität Bochum, Horst Görtz Institute for IT Security.
[19] V. I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In Soviet Physics Doklady, vol. 10 of no. 8, pp. 707–710.
[20] V. G. Li, M. Dunn, P. Pearce, D. McCoy, G. M. Voelker, and S. Savage. 2019. Reading the tea leaves: A comparative analysis of threat intelligence. In Proceedings of the USENIX Security Symposium. 851–867.
[21] L. B. Metcalf, D. Ruef, and J. M. Spring. 2017. Open-source measurement of fast-flux networks while considering domain-name parking. In Proceedings of the Conference on Learning from Authoritative Security Experiment Results (LASER’17). USENIX, 13–24.
[22] L. B. Metcalf and J. M. Spring. 2015. Blacklist ecosystem analysis: Spanning Jan 2012 to Jun 2014. In Proceedings of the 2nd ACM Workshop on Information Sharing and Collaborative Security. 13–22.
[23] Mozilla Foundation. 2019. Public suffix list. Retrieved from https://publicsuffix.org.
[24] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman. 2018. Sok: Security and privacy in machine learning. In Proceedings of the European Symposium on Security and Privacy. IEEE, 399–414.
[26] D. Plohmann, K. Yakdan, M. Klatt, J. Bader, and E. Gerhards-Padilla. 2016. A comprehensive measurement study of domain generating malware. In Proceedings of the USENIX Security Symposium. 263–278. Retrieved from https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/plohmann.

[27] P. Porras, H. Saidi, and V. Vegeswaran. 2009. Conficker C analysis. Technical report, SRI International. Retrieved from http://www.csl.sri.com/users/vinod/papers/Conficker/addendumC/index.html.

[28] U. Shamir. 2016. dganalyzer try to detect domain generation algorithms (DGA). Retrieved from https://github.com/udishamir/dganalyzer.

[29] C. Shannon. 1951. Prediction and entropy of printed english. Bell Syst. Tech. J. 30, 1 (1951), 50–64.

[30] A. K. Sood and R. J. Enbody. 2013. Crimeware-as-a-service: A survey of commoditized crimeware in the underground market. Int. J. Crit. Infrastruct. Protect. 6, 1 (2013), 28–38.

[31] J. M. Spring. 2013. Modeling malicious domain name take-down dynamics: Why eCrime pays. In Proceedings of the eCrime Researchers Summit (eCRS’13). IEEE.

[32] J. M. Spring, J. Fallon, A. Galyardt, A. Horneman, L. Metcalf, and E. Stoner. 2019. Machine Learning in Cybersecurity: A Guide. Technical report CMU/SEI-2019-TR-005, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. Retrieved from http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=633583.

[33] J. M. Spring, L. Metcalf, and R. Weaver. 2015. Domain blacklist ecosystem—A case study. Retrieved from https://insights.sei.cmu.edu/certi/2015/06/domain-blacklist-ecosystem-a-case-study.html.

[34] E. Stalman. 2017. DNS analysis scripts. Retrieved from https://github.com/staldradafastfluxanalysis.

[35] E. Stalmans and B. Irwin. 2011. A framework for dns based detection and mitigation of malware infections on a network. In Proceedings of the Conference on Information Security for South Africa. IEEE. DOI: 10.1109/ISSA.2011.6027531.

[36] J. Szurdi, B. Kocso, G. Cseh, J. M. Spring, M. Felegyhazi, and C. Kanich. 2014. The long tail of typosquatting domain names. In Proceedings of the 23rd USENIX Security Symposium. USENIX Association, 191–206.

[37] Verizon. 2015. Data Breach Investigations Report (DBIR). Technical report. Retrieved from http://www.verizonenterprise.com/DBIR/2015/.

[38] R. Vinayakumar, K. P. Soman, P. Poornachandran, M. Alazab, and A. Jolfaei. 2019. DBD: Deep learning DGA-based botnet detection. In Deep Learning Applications for Cyber Security, M. Alazab and M. Tang (eds.). Springer International, Cham, 127–149.

[39] R. Vinayakumar, K. Soman, P. Poornachandran, M. Alazab, and S. Thampi. 2019. Amritadga: A comprehensive data set for domain generation algorithms (dga) based domain name detection systems and application of deep learning. In Big Data Recommender Systems. Volume 2, 455–485.

[40] Z. Whittaker. 2015. Panda antivirus mistakenly flags itself as malware, bricks PCs. Retrieved from https://www.zdnet.com/article/panda-antivirus-mistakenly-flags-itself-as-malware-breaks-pcs/.

[41] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant. 2016. Predicting domain generation algorithms with long short-term memory networks. Retrieved from http://arxiv.org/abs/1611.00791.

[42] S. Wu and U. Manber. 1992. Fast text searching allowing errors. Commun. ACM 35, 10 (1992), 83–92.

[43] B. Wylie. 2014. dga_detection. (Sept. 2014). Retrieved from https://github.com/SuperCowPowers/data_hacking/tree/master/dga_detection.

[44] S. Yadav, A. K. K. Reddy, A. N. Reddy, and S. Ranjan. 2010. Detecting algorithmically generated malicious domain names. In Proceedings of the Internet Measurement Conference. ACM, 48–61.

[45] M. Zago, M. G. Pérez, and G. M. Pérez. 2019. Scalable detection of botnets based on DGA. Soft Comput. (2019), 1–21.

[46] M. Zago, M. G. Pérez, and G. M. Pérez. 2020. Umudga: A dataset for profiling dga-based botnet. Comput. Secur. 92 (2020), 101719.

[47] H. Zhang, M. Gharaibeh, S. Thanasoulas, and C. Papadopoulos. 2016. Botdigger: Detecting dga bots in a single network. In Traffic Monitoring and Analysis. IFIP, Louvain la Neuve, Belgium.

Received September 2019; revised September 2020; accepted September 2020