SDBF: Smart DNS Brute-Forcer

Cynthia Wagner, Jérôme François, Radu State, Thomas Engel, Gérard Wagener, Alexandre Dulaunoy

SDBF: Smart DNS Brute-Forcer

Cynthia Wagner, Jérôme François, Radu State, Thomas Engel
University of Luxembourg
SnT - Interdisciplinary Centre for Security, Reliability and Trust
L-1359 Luxembourg
Email: firstname.lastname@uni.lu

Gerard Wagener, Alexandre Dulaunoy
CIRCL
Computer Incident Response Center Luxembourg
Luxembourg
Email: firstname.lastname@circl.lu

***Abstract*—The structure of the domain name is highly relevant for providing insights into the management, organization and operation of a given enterprise. Security assessment and network penetration testing are using information sourced from the DNS service in order to map the network, perform reconnaissance tasks, identify services and target individual hosts. Tracking the domain names used by popular Botnets is another major application that needs to undercover their underlying DNS structure.**

Current approaches for this purpose are limited to simplistic brute force scanning or reverse DNS, but these are unreliable. Brute force attacks depend of a huge list of known words and thus, will not work against unknown names, while reverse DNS is not always setup or properly configured. In this paper, we address the issue of fast and efficient generation of DNS names and describe practical experiences against real world large scale DNS names. Our approach is based on techniques derived from natural language modeling and leverage Markov Chain Models in order to build the first DNS scanner (SDBF) that is leveraging both, training and advanced language modeling approaches.

I. Introduction

DNS means **D**omain **N**ame **S**ystem and represents a hierarchical naming system that translates domain names from human readable form into IP addresses for computers/services connected to the Internet and back. This not only makes DNS indispensable for surfing the net, but it also fills the gap as interface between human and machine. This also makes DNS attractive for attackers. For example, DNS probing is an important task in the reconnaissance phase, where an attacker collects information. DNS reveals valuable information about potential targets, as for example infrastructure information, MX or NS records, etc. This valuable information can serve as an attack point for attacks, as for example the referencing to hostile zone transfers, which should not be enabled in duly configured networks, therefore probing needs to be done.

In this paper, a new approach for the generation of new DNS names for probing purposes is presented. The smart DNS Brute-Forcer (SDBF) tool relies on a natural language processing method, called n-gram model, that uses a Markov chain to synthesize new DNS names. These newly created DNS names are validated on the net and evaluated by comparing the tool with others.

The paper is organized as follows: The SDBF model is described in section II, where the main features and modules are presented. Furthermore, this section describes how to generate new DNS names by referring to the n-gram model with its Markov chain. In section III, the experimental results are discussed and the performance evaluated. Section IV includes relevant research work for this domain and section V describes possible future work and presents the conclusions.

II. The architecture

The SDBF tool is composed of two distinct modules, the Processor and the Generator. The Processor generates statistics and extracts features from the passive DNS input. It is responsible for the generation of the n-gram probability distribution (see section II-B). The Generator module is used to compose

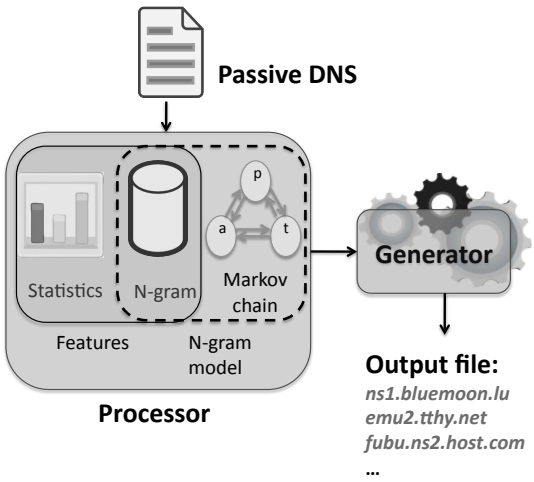


Fig. 1: Architecture of Smart DNS Brute-Forcer (SDBF)

new DNS names by using the n-gram probabilities. The newly created DNS names are then validated by performing DNS probing. Figure 1 shows the architecture of SDBF.

A. The Features and Statistical Parameters

To generate new DNS names with SDBF, some features have been identified. These features mainly are distributions for different linguistic attributes, as for example the character distribution. Passive DNS files, provided by a local network operator have been used to extract these features. The DNS file has a set of DNS names $N = \{n_1, ..., n_P\}$, a set of characters $C = \{c_1, ..., c_M\}$, a set of n-grams starting with a given character, for example x, $G_x = \{x_1, ..., x_T\}$ and a set of domain word levels $L = \{l_1, ..., l_S\}$, where $\#L$ has been

set to 4. In this paper the cardinality of a set is denoted by $\#$. We define,

- $\#wlen_n$: the number of DNS names having n words
- $\#len_{i,j}$: the number of words of the i^{th} level (with $i \in L$) having j characters
- $\#firstchar_{i,j}$: the number of words at the i^{th} level (with $i \in L$) beginning with character $j \in C$
- $\#ngram_{i,j,k}$: the number of times that character $j \in C$ is followed by character $k \in C$ at the i^{th} level and $i \in L$

The following list regroups different features that are extracted by the Processor,

- The number of words in a DNS name, $distwlen(x = k) = \frac{\#wlen_k}{\sum_j \#wlen_j}$
- The distribution for the word-length expressed in characters for a level l, $distlen_i$, can be defined as the length of a domain word of level i in a DNS name, $dist_i(x = j) = \frac{\#wlen_j}{\sum_k \#wlen_k}$
- The distribution $distfirstchar_i$ for the most occurring first character $firstchar_i$ per word of level i, $distfirstchar_i(x = j) = \frac{\#firstchar_{i,j}}{\sum_k \#firstchar_{i,j}}$
- The distribution of n-grams, called $ngram$ is generated from the n-grams extracted from the passive DNS file. A detailed description for the n-gram model is given in sub-section II-B.
  The distribution of n-grams can be defined as , $ngram_{i,j}(x = k) = \frac{\#ngram_{i,j,k}}{\sum_P ngram_{i,j,p}}$.

Besides these features, some statistical evaluations are made to complete the data set and model evaluation. This includes,

- The number of DNS entries in a file, n_{all}
- The average DNS name length in characters, including numbers and special characters (i.e. -, /, 2, etc.)
- The character frequencies for each character, number or special character of a DNS name

B. N-gram Model

In natural language processing, n-grams [1] are sequences of *n* consecutive characters of a string extracted from a corpus (a collection of texts, sentences, etc.[1]), with length *n*=1, 2, 3, 4,.... A n-gram of length 1 is called a unigram, n = 2 a bigram,... . By a first manual investigation of DNS names, the n-gram approach in this paper not only considers letters, but also special characters and numbers, as domain names cannot be compared with simple text. To illustrate an example of n-grams, please consider the following example, `test1.ex2ample.net`,

```
n = 1,   t, e, s, t, 1, ...
n = 2,   te, es, st, t1, ...
n = 3,   tes, est, st1, ...
```

In a first step, a passive DNS file is used as training to generated the n-gram frequencies by applying the formulas from section II-A. The composition of new DNS names by using the n-gram model which includes a Markov chain. Referring to the notation of [2], [1], suppose $X = X_1, ..., X_m$, as a set of random variables, then a Markov chain can be defined as a set of states S=$\{s_1$,s_2,...,$s_r\}$, where a process starts in a given state and continuously moves from one state to another or stays in the same state, and a move is called step. A Markov chain respects the Markov properties[1], which say that future states only depend on current states (history). A state changes into another one with a certain probability or remains in the same state with a certain probability, this is also known as transition probability. These transition probabilities are taken from the n-gram distribution. If a Markov chain is in state s_i, it changes into state s_j with probability p_{ij} or stays in the same state with probability p_{ii}. Furthermore, an initial probability distribution about S marks the Markov chain start state. An example of a state diagram for a Markov chain is given in Figure 2. The n-gram transition probabilities from one state to an other other are represented. Here, the probability that 'u' is followed by 'n' is 0.4, whereas the probability of having a 'u' followed by a letter 'i' is 0.6. The probability that an 'i' is followed by another 'i' is 0.2.

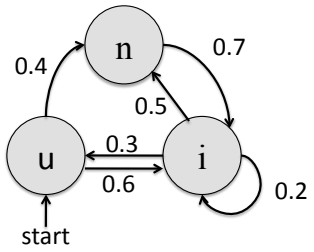


Fig. 2: Markov chain for n-grams

C. The Generator Module

The generator module in Figure 3 is responsible for the generation of new DNS names. In a first step, the number of domain words is generated, either by triggering a random number for the number of words in a DNS name or by using a custom number of words, see Figure 3(1). For each generated word, the character length is set by the user or randomly generated using the distribution $distwlenl_i$, being the word length distribution for a word level i, see (2). For each domain word, the first character is randomly selected by using the first character distribution, see (3). By taking the Markov probability transition matrix that has been generated from the n-gram model and the generated word lengths as input, words can be created by applying the most likely successive n-grams (based on the probability distribution). The user may also define a subpart of the DNS name, see (2'). For example in Figure 3, a DNS name of 3 words has to be created and the user has set up the second level as `uni`. The first level is automatically generated as `snt`. Adding a third level by using one of the previous methods then gives the full domain name `snt.uni.lu`. Thus, the user may easily probe domain names matching a regular expression like `ns.*.lu`.

[1]Markov properties following [1] notation, X is a Markov chain if Limited horizon: $P(X_{t+1} = s_k|X_1, ..., X_t) = P(X_{t+1} = s_k|X_t)$ and stationary : $= P(X_2 = s_k|X_1)$

To make the creation of DNS names more flexible, all the distributions consider a factor ϵ. For instance, even if the learning database does not contain any first level word starting with a `z`, this still can be generated, while introducing this small probability factor ϵ. SDBF can be run in parallel,

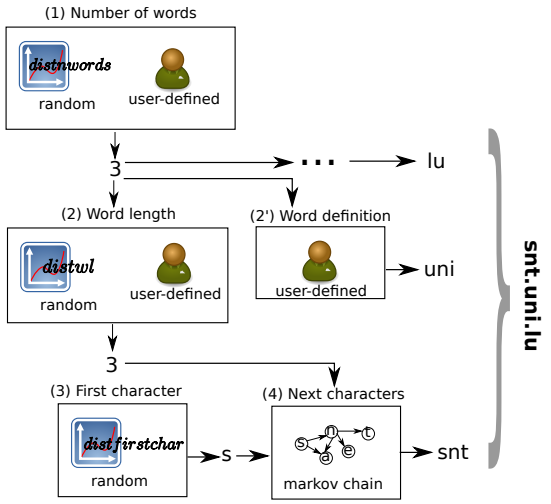


Fig. 3: Generator module

probing from multiple locations without probing the same DNS name multiple times. Moreover, language specificities might be discarded, since participating machines can be located in different countries, having each one their own local feature database. In addition, large enterprise networks deploy multiple authoritative servers for reliability reason. By performing parallel and iterative queries to this server from multiple internal locations, divergences can be revealed, which may indicate configuration errors.

III. Experimental Results

A. Learning Data

The learning data set that has been used has been provided from a local operator. This data set describes approximately 1 hour of passive DNS monitoring activity that has been captured between Resolver and Authoritative server. The table below regroups the principal characteristics of the learning data set,

Duration	60 min
Size	257 MB
Number of all domain names	1 M
Number of unique domain names	689 331
Avg. domain name length in characters	15.80
Avg. number of words	3
Size of alphabet	46
Most occurring first character	e

TABLE I: Characteristics for the learning data set

B. Validation

As SDBF aims to discover hostnames, the experimental evaluation considers $\#D$, the number of names that have been discovered on a probed domain. Since these numbers are highly dependent of the probed domain, the order of magnitude may vary a lot between two domains. Thus, this should be expressed as a ratio regarding the total number of real names that should have been discovered. Practically, getting such information is impossible.

The performance assessment is based on the comparison with other existing tools: Fierce [3] and DNSenum [4], both included in Backtrack [5], a linux distribution that is widely used by security experts for penetration testing. In [5], DNS probing is done by applying a dictionary of common machine names. DNSenum includes a large dictionary of 266 930 entries, whereas Fierce only has 1 895 entries. SDBF is not limited in its amount of names to generate. Thus, the value was set to 280 000 in order to have a similar value than DNSenum including a margin to see, if the testing of more hostnames is still efficient. Some other techniques are like Google, scrapping for getting more information about living hosts in a domain. Since, such techniques are out of scope for pure DNS probing, this kind of application has been discarded.

The SDBF validations assume the number of DNS names discovered by each tool: $\#D_{sdbf}$, $\#D_{dnsenum}$ and $\#D_{fierce}$. This can also be expressed as the ratio of the maximal number of names that any tool can discover,

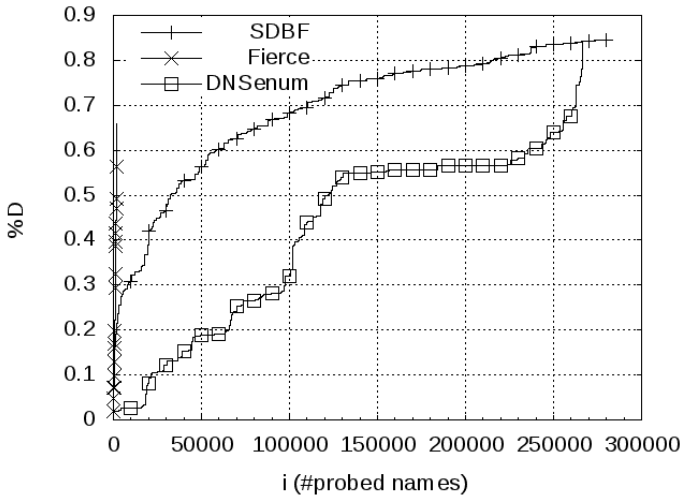
$$\%D_{sdbf} = \frac{\#D_{sdbf}}{max(\#D_{sdbf}, \#D_{fierce}, \#D_{dnsenum})} \quad (1)$$

However, this experiment was faced to virtual hosting, where the same host may match different names. This allows a single host to host multiple services on the same port. A standard example is a web server, where the DNS names differentiate the websites and the user who wants to access (see [6] for more details). A simple solution would be to take only different IP addresses into account, but this seems too restrictive, as it is important to detect different services, even if they are executed on the same machine. However, probing certain domains with different hostnames always returned positive results and in many cases the public web server. But these cases are not considered relevant and thus, IP addresses are considered. Otherwise, domain names are kept. The experiments focus on `A` requests, but SDBF can also be configured such that it can use other kinds of requests.

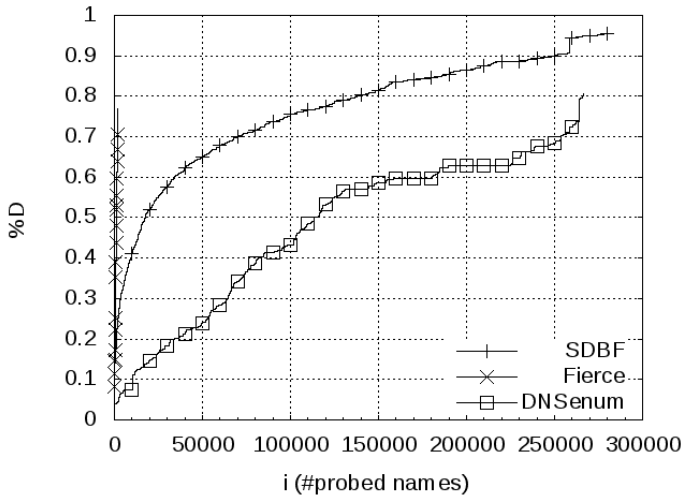
C. Hostname Probing

This first set of experiments aims to identify all hosts for a given domain. For example, one targeted domain is the university domain, `uni.lu`. The goal is to detect names matching the regular expression `*.uni.lu` like `www.uni.lu`. Furthermore, SDBF will also probe other subdomains and generate requests like `www.snt.uni.lu`, since the learning data behind includes domain names up to 4 words (see III-A.

Table II shows the number of discovered hosts for a probed domain name inspecting the number of items $\#D$ and the rate $\%D$. It can be seen in Table II that it holds first, some *local* domains from Luxembourg and second, domains located



(a) *Luxembourg* domains



(b) *World* domains

Fig. 4: Speed of name discovery

		SDBF		Fierce		DNSenum	
		#D	%D	#D	%D	#D	%D
Local	pt.lu	**71**	**1.00**	42	0.59	49	0.69
	uni.lu	57	0.85	22	0.33	**67**	**1.00**
	circl.lu	**17**	**1.00**	6	0.35	9	0.53
	vdl.lu	15	0.94	11	0.69	**16**	**1.00**
	wort.lu	4	0.50	**8**	**1.00**	7	0.88
	rtl.lu	14	0.78	**18**	**1.00**	17	0.94
World	google.com	69	0.82	**84**	**1.00**	83	0.99
	facebook.com	**101**	**1.00**	42	0.42	71	0.70
	youtube.com	**55**	**1.00**	49	0.89	35	0.64
	yahoo.com	**145**	**1.00**	110	0.76	138	0.95
	blogspot.com	**3**	**1.00**	**3**	**1.00**	**3**	**1.00**
	baidu.com	**409**	**1.00**	178	0.44	238	0.58
	wikipedia.org	**282**	**1.00**	143	0.51	139	0.49
	live.com	48	0.83	53	0.91	**58**	**1.00**
	twitter.com	31	0.94	**33**	**1.00**	30	0.91

TABLE II: Efficiency in discovering valid names (Bold font indicates the best tool for each domain)

around the world, called *world*. SDBF is the most efficient tool, since it has discovered the maximal number of valid names with a permanent ratio over 50%. Assuming the 75% of best results, the rate is at least 85% although for Fierce and DNSenum, this value drops to 69% respectively 51%.

SDBF seems more appropiate for probing worldwide than local based domains, even if it was trained using a local DNS capture. This may be due to the fact that *world* domains have been extracted from the top ranking of the Alexa web-site [7], leading to probe famous large domains with many hosts in the background, whereas *local* domains in Luxembourg are small with fewer hosts. Generating various DNS names, like with SDBF, is efficient.

D. Speed of Discovery

SDBF may generate infinitively new DNS names to probe and so, has greater chance to discover more hosts than ordinary dictionary based tools. This section analyses discovery process speed that can be defined as the number of valid probed names after i generated ones: $\#D^i$. This value can be expressed for a single tool from which the rate can be derived,

$$\%D^i_{sdbf} = \frac{\#D^i_{sdbf}}{max(\#D^i_{sdbf}, \#D^i_{fierce}, \#D^i_{dnsenum})}$$

where the notation $\#(x)$ is the cardinality of a set x.

This metric is plotted in Figure 4(a) and 4(b), where the average over all probed domains in Luxembourg and worldwide have been computed. Fierce is very fast in finding hosts, but since its dictionary is very small (1895 hostnames), it may miss less common names. The other tools have a lower discovery speed, but match more names. Comparing DNSenum and SDBF , the latter is always faster, except at the end of the curve where DNSenum is better. In fact, dictionaries are used in alphabetical order (as done usually) and the slope changes in the DNSenum curves mean that valid names exist in close alphabetical position. For example, the last slope change is mainly due to the following names `www`,`www2`, `web`, etc.

E. Tool Complementarity

In the previous experiment, the evaluation is based on the number of probed DNS names that were successful. However, sometimes good results can be achieved with a well designed dictionary. However, it may lead to probe only common names such as web, ftp or mail server. SDBF exceeds this limitation and can be considered as a complementary tool. Therefore, the next experiment evaluates the complementarity by counting the number of successful probed names, which have not been seen by the other tools. Three sets based on the previously defined denominations are defined,

- S_{sdbf}: names discovered by SDBF
- $S_{dnsenum}$: names discovered by DNSenum

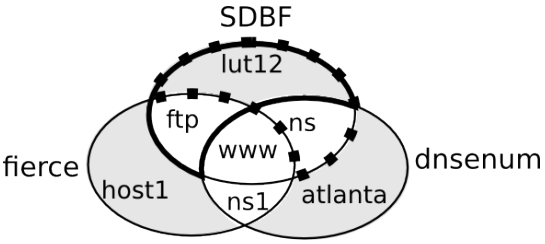


Fig. 5: Complementarity of tools – Shaded areas represents the uniqueness metric of the tools, dotted and bold lines delimit the complementarity of SDBF against DNSenum and Fierce separately

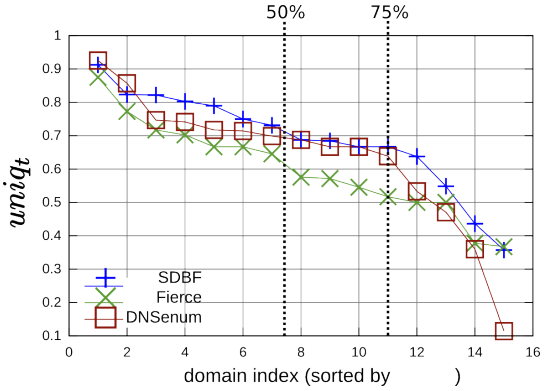


Fig. 6: Uniqueness of discovered valid names – The index represents the domains sorted by $uniq_t$

- S_{fierce}: names discovered by Fierce

Figure 5 is a illustrative example. The uniqueness metric for a tool can be defined as the ratio between the number of successfully probed names and the total number of generated names. This metric is represented by grey shaded areas in the figure and is formally defined for each tool $t \in T = \{sdbf, dnsenum, fierce\}$ as:

$$uniq_t = \frac{\#(S_t - \cup_{x in T, x \neq r} S_x)}{\#(S_t)} \quad (2)$$

To strengthen the validation, the SDBF complementarity against the other tools is considered (see Figure 5) by computing the following metric:

$$uniq_{t,y} = \frac{\#(S_t - S_y)}{\#(S_t)} \quad (3)$$

In order to focus on the evaluation of SDBF, t is always fixed to SDBF as shown in equation 3.

Figure 6 summarizes the results in a comprehensive manner. The x-axis represents domains sorted by descending order regarding $uniq_t$, but are expressed as an index, since the order may be different for each tool. Hence, the first index may not represent the same domain, but here, the goal is not to compare the tools per domain by figuring out the uniqueness of the discovered. From a general point of view, the tools have similar performances, but SDBF has a little advantage by having a higher average ratio (0.69). Since the index is sorted,

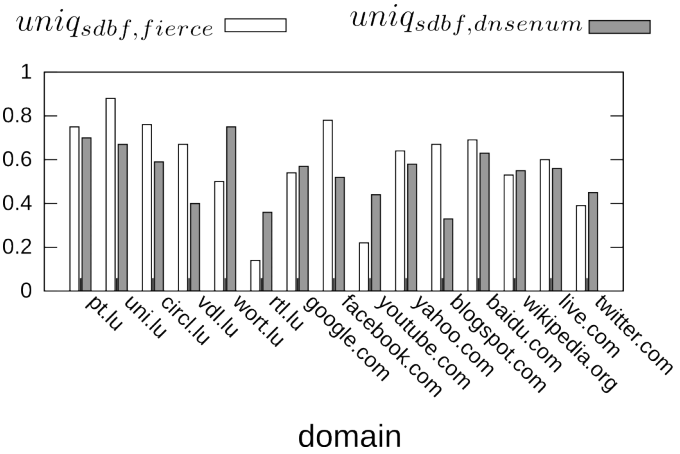


Fig. 7: Complementarity of SDBF against other tools $uniq_{sdbf,y}$

Figure 6 shows the 50^{th} and 75^{th} percentile by a vertical line. Hence, the $uniq_{sdbf}$ value for the 75% in SDBF (0.67) is higher than for Fierce (0.52) and DNSEnum (0.54). This means that SDBF is able to explore a large variety of DNS names, which have not been discovered by the other tools. To be brief, all tested tools are very complementary.

Figure 7 shows the uniqueness of values discovered by SDBF against each tool individually ($uniq_{sdbf,y}$). DNS names find out by SDBF seems more unique compared with Fierce that DNSenum. However, this small bias is mainly due to a very small dictionary. Furthermore, this is not always the case especially for `rtl.lu` and `facebook.com` where $uniq_{sdbf,fierce}$ is quite lower than $uniq_{sdbf,dnsenum}$. Such examples prove that even SDBF uses a random process to generated domain names and the learning phase is quite efficient to generate new valid names, as those that have been fed into expert dictionaries.

To summarize the previous experiments, SDBF is able to discover more DNS names and the majority of these are not discovered by dictionary based approaches. This justifies the cost of SDBF, even if the generation of many requests is needed. For instance, the experiments require the generation of 280 000 names, unlike Fierce is limited to 1 895 names. Thus, the right trade-off has to be made between the names to discover and the risk to be filtered, since SDBF is quite noisy from a network point of view. To avoid this issue, distributing the scanning over multiple machines can be performed, as mentioned in II-C, but another alternative could be to use caching servers. By leveraging iterative queries, SDBF can directly probe the cache of a non-authoritative server, hiding itself from the targeted domain. For a better efficiency, such an approach has to query highly loaded server like for example, the public DNS of Google [8].

F. Domain Name Probing

Usual tools based on dictionaries are designed to probe all hostnames of a domain, like in the previous experiments. For instance, they may generate FQDNs matching a regular

expression like `*.uni.lu`. SDBF is able to match an expression like `ns.*.lu` (looking for name servers in `.lu` domain), since it processes each word separately and the user can set the name structure. Below, some examples of invalid probed domain names for the domain `*.pt.lu` are given.

```
gbnsy.1bl.pt.lu
sa174fz7246fof35to.du.pt.lu
nes.pt.lu
```

Some examples for valid domain names are,

```
www.pt.lu
roma.pt.lu
mmail.pt.lu
```

In an additional experiment, FTP [9] and name servers for domains in Luxembourg have been probed. This matches the following expression: `ns.*.lu` and `ftp.*.lu`. Even if this experiment does not include all possible existing names, finally 30 name servers and 706 FTP servers have been discovered.

IV. Related Work

In the research area of DNS analysis, papers can be mainly classified into two different categories, surveys and papers implementing solutions of most different kind. The first category of papers relevant for this research area are the surveys that have been made, as [10], [11]. These describe possible attack schemes as suspicious port numbers, mass-mailing, spam, fast flux, etc. and relevant countermeasures. The second category of papers applies approaches on statistical evaluations or present new methods to redesign DNS [12], but only a few papers treat about the knowledge on domain names,[13], [14].

For example in article [15], the authors want to detect various fast poisoning attacks which manipulate resolution caches through different methods of blind off-path guessing of transaction components, which are used for DNS message integrity. In this kind of attack, clients are redirected to new different IP addresses of harmful sites. To counter-fight this attack, in [15] it is referred to statistical evaluations with whitelists and different kind of classifiers, as k-nearest neighbors or support vector machines to detect anomalies in RR data. In [16], the authors describe the deployment of a university monitoring tool on the local network and monitored the network activities. [16] describes different possible DNS anomalies and as evaluation tool different statistics are drawn, for example by simply studying typographical errors in DNS entries or the amount of changes in IP addresses for fast-flux domains. In [17], a large-scale passive DNS tool is described. Exposure decribes a large-scale passive DNS analysis tool that relies on a selection of 15 different features. In [17], the focus is not on one special kind of anomaly, but the detection of various malicious behaviours. The defined features are time-based for long period analysis, or the calculation of the euclidean distance between entries in order to detect abrupt changes. They discovered for example that short-lived domains have only two sudden behaviour changes and long-lived domains have multiple behavioural changes. In their experimentation, they used a classifier which was trained during a week and then they were only able to test their tool. The more, the false positive rate was estimated by performing different experiments as manually checking domains on Google. In paper [18], the autors majorily describe how to analyze top-level DNS domains listed by the VerSign Servers. For the analyses they use a similiarity metric, the Jaccard Index to compare sets of of aggregated (/24) IP-addresses. To analyze Network-wide patterns, they simply calculate the average of Jaccard Indexes and by this can regroup domains into different classes, as for example phishing, spam etc. The paper shows that malicious domains show more variance in their network patterns that look them up.

It can be observed that a lot of research is mainly focused on the evaluation of anomalies respectively attacks on DNS. In this paper, it is more investigated on the visibility of domains that are on the network. Therefore, another focus is to look at the domain names themselves. A recent observed trend is the use of natural language processing techniques in forensic and security research. An example is [13], where domain names are automatically. They refer to a syllable based algorithm to generated passwords or usernames. The main difference to our paper is that they mainly generate full words, in our approach the n-gram approach is used such that characters can be generated into words by using a probability distribution. In [19], the technique presented in [13] is used to generate domain names. With aid of statistical approaches, as Kulback-Leibler divergence or edit distance measures, domain names generated by well-known Botnets for domain fluxing should be detected. In [14], different approaches for extracting DNS meta-data in reconnaissance phase are presented. Therefore an inference algorithms, as Dictionary attack or Brute forcing are applied to generate names and send to the DNS resolver. Another interesting work for security aspects is the paper of [20], where it has been referred to probabilistic context-free grammar for generating rules to crack passwords.

V. Conclusion

In this paper, a new approach for the smart generation of new DNS names has been introduced that referred to a natural language processing method, the n-gram model. The advantage of SDBF is that it is not limited to the knowledge of dictionaries, as it is common for this kind of expert tools. The experiments show that SDBF is able to discover more names than dictionary-based tools. This can be explained by the fact, that the n-gram model does not need to generate existing words, but also can generated simple sequences of characters including numbers and special characters for DNS names.

Acknowledgments

We acknowledge Prof. T. Duhautpas from Restena Luxembourg for his counsel. This project is partially funded by the FNR Luxembourg.

REFERENCES

[1] C. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. Cambridge, MA, USA: MIT Press, 1999.

[2] C.M.Grinstead and J.N.Snell, *Introduction to Probability*. American Mathematical Society, 1997.

[3] "Fierce," http://ha.ckers.org/fierce/.

[4] "Dnsenum," http://code.google.com/p/dnsenum/.

[5] "Backtrack," http://www.backtrack-linux.org/.

[6] "Virtual host," http://httpd.apache.org/docs/2.0/vhosts/.

[7] "Alexa website," http://www.alexa.com.

[8] "Google Public DNS," accessed on 08/22/11. [Online]. Available: http://code.google.com/speed/public-dns/index.html

[9] "Ftp," http://tools.ietf.org/html/rfc959.

[10] D. David, P. Niels, L. C. L., and L. Wenke, "Corrupted dns resolution paths: The rise of a malicious resolution authority," in *Proceedings of NDSS'08*, 2008.

[11] H. van Heide and N. Barendregt, "Dns anomaly detection," 2011.

[12] V. Ramasubramanian and E. Sirer, "The design and implementation of a next generation name service for the internet," *SIGCOMM Comput. Commun. Rev.*, vol. 34, pp. 331–342, August 2004.

[13] H. Crawford and J. Aycock, "Kwyjibo: automatic domain name generation," *Software Practice and Experience*, vol. 38, pp. 1561–1567, November 2008.

[14] A. da Silveira and N. Garcia, "Algorithms for extraction and visualization of metadata from domain name server records," in *Third International Conference on Advances in Mesh Networks*. IEEE, 2010, pp. 81–85.

[15] M. Antonakakis, D. Dagon, X. Luo, R. Perdisci, W. Lee, and J. Bellmor, "A centralized monitoring infrastructure for improving dns security," in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, vol. 6307, pp. 18–37.

[16] B. Zdrnja, N. Brownlee, and D. Wessels, "Passive monitoring of dns anomalies," in *Proceedings of the 4th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. DIMVA '07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 129–139.

[17] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "Exposure : Finding malicious domains using passive dns analysis," p. 17, 2011. [Online]. Available: http://iseclab.org/papers/bilge-ndss11.pdf

[18] S. Hao, N. Feamster, and R.Pandrangi, "An internet wide view into DNS lookup patterns," School of Computer Science, Georgia Tech, Tech. Rep., june 2010. [Online]. Available: http://labs.verisign.com/projects/malicious-domain-names.html

[19] S. Yadav, A. K. K. Reddy, A. N. Reddy, and S. Ranjan, "Detecting algorithmically generated malicious domain names," in *Proceedings of the 10th annual conference on Internet measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 48–61.

[20] M. Weir, S. Aggarwal, B. d. Medeiros, and B. Glodek, "Password cracking using probabilistic context-free grammars," in *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 391–405.