Semantic Exploration of DNS

Samuel Marchal, Jérôme François, Cynthia Wagner, and Thomas Engel

SnT - University of Luxembourg, Luxembourg,
firstname.lastame@uni.lu

Abstract. The DNS structure discloses useful information about the organization and the operation of an enterprise network, which can be used for designing attacks as well as monitoring domains supporting malicious activities. Thus, this paper introduces a new method for exploring the DNS domains. Although our previous work described a tool to generate existing DNS names accurately in order to probe a domain automatically, the approach is extended by leveraging semantic analysis of domain names. In particular, the semantic distributional similarity and relatedness of sub-domains are considered as well as sequential patterns. The evaluation shows that the discovery is highly improved while the overhead remains low, comparing with non semantic DNS probing tools including ours and others.

1 Introduction

DNS (Domain Name System) [15] is critical for the well functioning of Internet as it is mainly used for locating a host in the Internet based on a human readable name. Service availability is improved by dynamic reallocation to another machine without changing the DNS name. However, this mechanism is also employed by attackers to improve the robustness and the efficiency of the attacks [18]. Hence, DNS has recently gained interest from the security community and especially the naming scheme for discovering malware hosting domains [18].

This paper focuses on DNS probing, *i.e.* guessing domains that are in use. This is an alternative to IP address scanning, which is fastidious and quite visible whereas DNS requests go through intermediate DNS servers, which hide the attackers. An attacker commonly uses dictionaries to probe existing domain names and aims to discover the networking organization, as well as potential vulnerable hosts. A common example is to check the hostnames of common services like FTP (File Transfer Protocol) or SSH (Secure Shell) by probing domains such as `ftp.example.com`. In [10], the authors show that DNS scanning allow to identify potentially vulnerable IPv6 addresses quicker than with a classical random IP scanning due to the large address space, they apply this technique to ease the spread of worms in IPv6 Internet.

Thus, penetration testing and security assessment are based on an initial recon by discovering subdomains and hosts. With a proper DNS configuration, this cannot be gathered directly and therefore requires brute-forcing. In this paper, the DNS brute forcing tool is semantically extended since we have observed that human based names usually follow semantic schemes. This includes the word semantic as well as numerical semantics (series of numbers) of DNS names.

The paper is organized as follows. Section 2 presents DNS. An overview of the semantic exploration system and SDBF (Smart DNS Brute-Forcer) [21] is given in section 3. Semantic extensions are covered in section 4. Our approach is assessed in section 5. Related work is presented in section 6 and conclusions are drawn in section 7.

2 DNS Background

To keep the paper self-contained, this is a short overview of DNS, but the reader may read [16, 15, 17] for further explanation.

The main objective of DNS is to provide a map between human readable and remainable names to IP addresses. The organization of DNS is hierarchical with a root server at the top and dedicated authoritative servers for each subdomain. Assuming the domain name `www.uni.lu`, `lu` is the top level domain (TLD) which is the parent of all `.lu` subdomains (second level domain) including `uni.lu`. The third level domain is `www.uni.lu`. When a user needs the IP address of `www.uni.lu`, the first step is to query a recursive DNS server, usually maintained by his operator. This server is responsible to find the host by iteratively querying the authoritative servers of the subdomains. So, it starts by asking a root server which replies back with the DNS server in charge of the `lu` domain. The recursive DNS server of the client can also contact it to know, which server is in charge of `uni.lu`. Finally, when `uni.lu` is queried, it returns the IP address of `www.uni.lu`, which is then forwarded to the client by the recursive server.

DNS messages are mainly composed of Resource Records (RRs), which refers to different types of resolution. For the most common one the type is `A` or `AAAA` for getting the IPv4 or IPv6 address corresponding to a domain name. `PTR` are defined to enable inverse resolution (IP address to name).

A DNS name uses a dotted format to separate several components, *i.e.* a sequence of labels. In this paper, $label_i$ refers to the i^{th} component, starting from the right. Thus, the top level domain is defined by $label_0$. For example, `www.uni.lu` has three labels: $label_0 = lu$, $label_1 = uni$, $label_2 = www$. Even if a recent extension allows non-ASCII characters [7], this paper doesn't consider them, as most of domains are still composed only of ASCII characters.

3 Exploration of DNS

3.1 System Overview

Our approach aims to automatically discover DNS names, and in particular, some subdomains of a domain by generating labels. Assuming a domain d, most of the current techniques rely on testing labels sequentially, l, stored in a dictionary (www, ns, ftp, smtp, etc. but also atlanta, boston, host, etc.) by concatenating the label l with the domain d to form a new subdomain $l.d$. In this paper, our prior tool SDBF [21] is used to generate new names after a learning stage. Samples are required to learn how valid labels of domain names look like. They

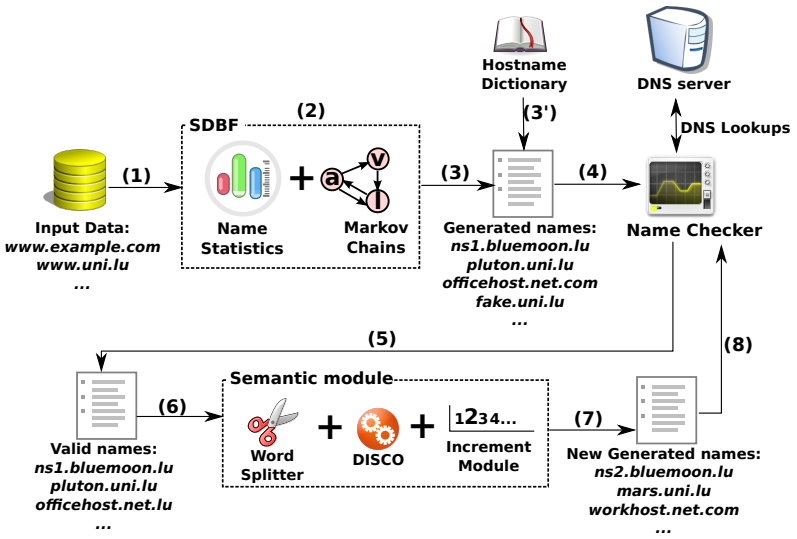


Fig. 1. System overview

are collected through a passive DNS platform [22], which consists in monitoring and storing requests sent and replies received by a recursive DNS server. In our case, only valid names *i.e.* with valid DNS replies, are stored in a database.

Two key ideas have emerged from observations we made during our personal experience, as well as by mining the passive DNS database:

- subdomains of the same domain are semantically related, in particular end hosts. For example, using planets, cities, countries, or character names of cartoons is a frequent habit of network administrators,
- a domain may present sequential patterns. For example, enumeration is a standard naming convention within a company or a university that leads to hostnames like room1-pc1, room1-pc2, room2-pc1, ns1, ns2, etc.

As shown in figure 1, the two main steps for discovering the DNS names are:

- the construction of an initial list of names using SDBF [21] (2) or a dictionary based-approach (3')
- the extension of that list by exploiting the semantics of names (5)-(8)

3.2 SDBF

Features The main features in SDBF are based on linguistic parameters. We assume an input list (1) of DNS names $N = \{n_1, ..., n_P\}$, a set of DNS label levels $L = \{l_1, ..., l_S\}$, a set of used characters $C = \{c_1, ..., c_M\}$ and a set of n-grams, $G_x = \{x_1, ..., x_T\}$.

The **statistical** features include: $\#wlen_n$ - the number of DNS names with n labels, $\#wlen_{i,j}$ - the number of labels of the i^{th} level (with $i \in L$) having

j characters, $\#firstchar_{i,j}$ - the number of labels at the i^{th} level (with $i \in L$) starting with character $j \in C$ and $\#ngram_{i,j,k}$ - the number of times that a character $j \in C$ is succeeded by $k \in C$ at the i^{th} level with $i \in L$.

These features are transformed into distributions as follows ((2) in figure 1):

- the distribution for domain lengths (in label levels):

$$distwlen(X = j) = \frac{\#wlen_j}{\sum_k \#wlen_k} \tag{1}$$

- the distribution of the lengths of labels (in number of characters) for a given level l:

$$distwlen_l(X = j) = \frac{\#wlen_{l,j}}{\sum_k \#wlen_{l,k}} \tag{2}$$

- the distribution of the first characters of labels for a given level l:

$$distfirstchar_l(X = j) = \frac{\#firstchar_{l,j}}{\sum_k \#firstchar_{l,k}} \tag{3}$$

- the N-gram distribution for a certain level l and a character c is given by:

$$ngram_{l,c}(X = i) = \frac{\#ngram_{l,c,i}}{\sum_k ngram_{l,c,k}} \tag{4}$$

N-gram Model: N-grams [14] are successive character sequences of length $n \in \mathbb{N}$, extracted from a string. For example, an n-gram with $n = 2$ is called *bigram*. Consider the following DNS name, `test.uni.lu`, bigrams can be: `te, es, st, un, ni, lu`... For generating the names of labels, the different estimated distributions are applied to a Markov chain. A Markov chain is defined for each label level, l, as a set of states $S=\{s_1,s_2,...,s_r\}$ representing the characters that have been observed at this level. The probability of the transition between the two nodes representing by the two characters c_i and c_j is equivalent to $ngram_{l,c_i}(X = c_j)$. By applying k steps, this model allows to generate a label of k characters.

An example for the n-gram model Markov chain is given in figure 2. This means, the probability that a character 'u' is followed by character 'n' is 0.4 and the probability that an 'i' is followed by another 'i' is only 0.2.

Name generation: Once the system is trained, SDBF can generate new names to probe, by first defining how long the new name should be in terms of number of labels. To achieve this, a random number following the distribution of number of labels, ($distwlen$), is generated. As SDBF is designed to be highly customizable, this value can also be set by the user. The same process is applied to determine the length of labels in characters for each label l to generate: $distwlen_l$. Again, the user can set the value. Finally, for a label with a length k, the first character will be generated following the distribution of the first characters corresponding

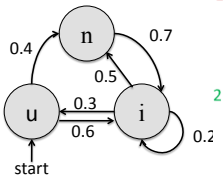


Fig. 2. Markov chain for n-grams

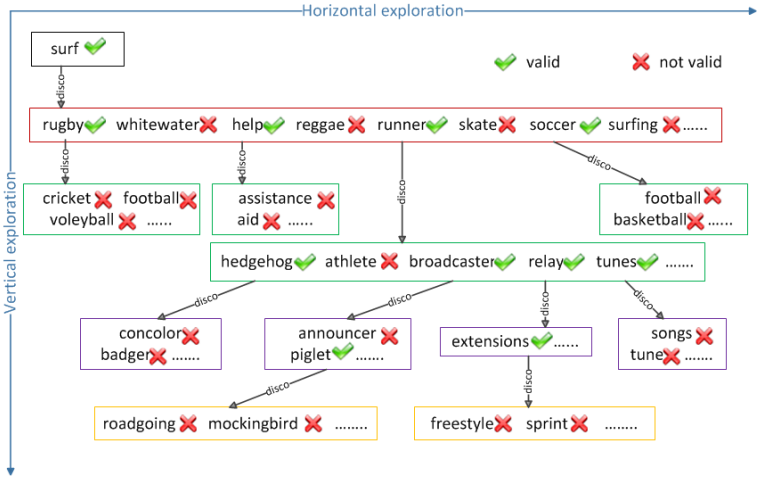


Fig. 3. Example semantic exploration from `surf.apple.com`

to the label level, $distfirstchar_l$, and the remaining $k-1$ characters are generated by applying the Markov Chain.

As the Markov chain is limited to a fixed set of transition, some transitions are not possible. For instance, if the bigram "sn" was never observed, the word "snt" could not be generated. To strengthen the discovery, we consider that a transition between any two pairs of characters is possible with a probility ϵ. For this, the other tansitions probabilities are slightly decreased to keep the sum of probabilities for outgoing transitions equal to one.

Because it is common usage to scan a domain, the user can set fixed parts for a domain. For example, the objective may be fixed to discover all domains following the `*.uni.lu` or `ns.*.lu` or `www.*.*`.

Once names are generated, (3) in figure 1, their existence is checked by the name checker (4), which makes a DNS query. This formally corresponds to a function $valid(D)$ returning the valid domains of a set D.

4 Semantic extension

As illustrated in figure 1, the semantic module takes as input a list of names, where the validity has been checked (5). The goal is to extend this list of discovered names by analyzing individual labels. There are two modules, DISCO and the incremental module that can be used individually or combined together, whereas the splitter module is an optional preprocessing step.

4.1 Similar names

The first semantic extension aims to discover names that are similar or related. These are distinct notions[4]. Similarity refers to words having a close meaning

(for example, notebook and laptop). Semantic relatedness refers to words sharing the same semantic field like mars and venus, which are different planets. As claimed by Kilgraff et. al [11], these usual notions impliy a manual analysis to establish relationships between words that limits its applicability and its extension to further language or semantic domain.

In this paper, we leverage DISCO (extracting DIStributionally related words using CO-occurrences) [12], which is based on an efficient and accurate method for approximating automatically (based on leanring samples) these two notions within one metric, called the similarity afterwards. DISCO considers the distance between two words within a window by defining $||w, r, w'||$, the number of times the word w' occur after r words after the word w, where $-3 \leq r \leq 3$. For example, table 1 represents windows centered on `services`. The window is moving along all the database samples to compute the counting that is transformed into frequencies, *i.e.* $f(w, r, w')$, by dividing by the total number of counted co-occurences for any $||w, r, w'||$.

Intuitively, two words w_1 and w_2 are considered similar, if both of them have many co-occurrences with the same words, in particular, if the positions the latter regarding w_1 and w_2 are similar. DISCO uses the following definition, initially proposed in [13]:

$$sim(w_1, w_2) = \frac{\sum_{(r,w) \in T(w_1) \cap T(w_2)} I(w_1, r, w) + I(w2, r, w)}{\sum_{(r,w) \in T(w_1)} I(w_1, r, w) + \sum_{(r,w) \in T(w_2)} I(w_2, r, w)} \quad (5)$$

where $I(w, r, w')$ is the mutual information between w and w' [9] and $T(w)$ all the pairs (r, w') where $I(w, r, w')$ is positive.

Assuming a domain d including the label l, the objective is to find similar labels l'. The exploration goes into two directions. The first one is the horizontal exploration, which may be adjusted by lim_h. This corresponds to select the most lim_h similar words from DISCO. This result is set into a new set of labels $Expl_H(l, lim_h)$ which are tested by the *Name Checker* (figure 1) by concatenating with unmodified labels (other levels). By this, a new set is obtained, denoted by $Valid(Expl_H(l, lim_h))$.

The second exploration examines the vertical dimension by looking for additional similar names starting from this new set. The limit of the vertical explo-

position	-3	-2	-1	0	+1	+2	+3
sample 1	a	client	uses	services	of	the	platform
sample 2	the	platform	provides	services	to	the	client

$\|\|services, -3, a\|\| = 1$	$\|\|services, -3, the\|\| = 1$
$\|\|services, -2, client\|\| = 1$	$\|\|services, -2, platform\|\| = 1$
$\|\|services, -1, uses\|\| = 1$	$\|\|services, -1, provides\|\| = 1$
$\|\|services, 1, of\|\| = 1$	$\|\|services, 1, to\|\| = 1$
$\|\|$**services,2,the**$\|\| =$ **2**	$\|\|services, 3, client\|\| = 1$
$\|\|services, 3, platform\|\| = 1$	

Table 1. Example of co-occurrence counting (2 windows centered on *services*)

ration is set by lim_v and is defined by repeating the previous process lim_v times with new discovered valid names:

$$Expl_V(l, lim_v) = \begin{cases} \emptyset & \text{if } lim_v = 0 \\ \bigcup_{l' \in Expl_H(l, lim_h)} Valid(Expl_H(l', lim_H)) & \text{if } lim_v = 1 \\ \bigcup_{l' \in Expl_V(l, lim_v - 1)} Valid(Expl_H(l', lim_H)) & \text{otherwise} \end{cases} \tag{6}$$

In order to reduce the search space only validated labels are considered for further extensions, as noticed by the use of $Valid$ in the equation (6). The vertical exploration stops once no new correct labels are found. So, lim_v does not need to be manually set, which improves the easy use of our tool.

The vertical exploration is actually recursive and highlighted in figure 1 by the loop (5)-(6)-(7)-(8). Figure 3 represents a subset of a real probing by starting from the label `surf`, the horizontal exploration reveals unsuccessful (surfing, skate) and successful (rugby, soccer...) labels. Then, the vertical exploration entails a horizontal extension for each of the latter.

4.2 Incremental discovery

In many cases, machines and services are replicated and/or respect a systematic naming scheme as for example `pc1`, `pc2`, etc. Assuming that one of them has been discovered, the others can be generated by finding out the numerical components and using the following heuristic: test all possible values (including ∅) for each individual digit. This limits the exploration to a number of the same or of smaller power of ten (0 to 9 will in the previous example). Preliminary experiments have shown that increasing the search range to bigger numbers does not improve the results while, the overhead highly increases.

4.3 Splitter

Labels of DNS names can be composed of several words like `linuxserver` or `linux-server`. Applying DISCO on such names cannot provide any results since it performs over single words. Therefore, the labels have to be divided automatically in advance. Using a list of separating characters, as for instance "-" is too restricted and our tools refer to the word segmentation method described in [20]. The process is recursive by successively dividing the label in 2 parts, to find the best combination, *i.e.* with the maximum probability, of the first word and the remaining part. Therefore, a label l is divided in 2 parts for each position i and the probability is computed:

$$P(l, i) = P_{word}(pre(l, i))P(post(l, i)) \tag{7}$$

where $pre(l, i)$ returns the substring of l composed of the first i characters and $sub(l, i)$ of the remaining part. $P_{word}(w)$ returns the probability of having the word W equivalent to its frequency in a database of text samples.

Additionally, the splitter modules can also discover the incremental part of a domain. A label like `computer23` is split as `computer` and `23` which is helpful for the first step of the incremental process (see previous subsection). This may also detect non numerical increments, as observed in our database (`servera`, `serverb`, etc.), which can be incremented afterwards using ASCII codes.

5 Evaluation

5.1 Methodology

Assuming a domain d, dictionary based techniques probe by iterating over a set of labels, l, to form the hostname $l.d$. In the current evaluation, SDBF is configured similarly and two dictionary-based tools are also tested: Fierce [1] and DNSenum [2]. Both are included in Backtrack [1], a Linux distribution designed for digital forensics and penetration testing. The dictionary from Fierce includes only 1 895 words, whereas the one from DNSenum includes 266 930 entries. Hence, SDBF was configured to generate as many labels as DNSenum. The reader should refer to [21] for an evaluation of these tools without semantic extension. The main result is that SDBF and Fierce provide the best results, but all of them are complementary, *i.e.* they do not find the same names.

Based on the discovered names, new ones are probed using the semantic extensions with one of the following strategies:

- Similar names (DISCO)
- Similar names (DISCO) + Splitter
- Similar names (DISCO) + Splitter + Incremental discovery

Except if mentioned, the last one is applied. The original databases provided with the semantic tools [12, 20], like Wikipedia [3], are used to train them.

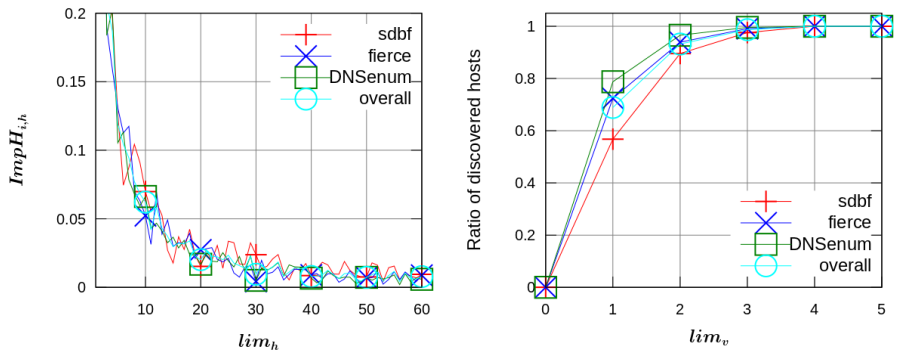
The targeted domains in our experiment are extracted from the top 50 websites ranked by Alexa (*www.alexa.com*), where only 19 domains have been selected such as `google.com`, `ebay.com`, `baidu.com`... This selection discards domains performing wildcarding *i.e.* these domains will always respond positively to DNS requests regardless of the query. Furthermore, similar domains with different TLD have also been discarded, since hostname results are similar in this case. For example, `google` has no less than twelve domain names with different TLDs in the top 50 Alexa. We also choose five *popular* domains from Luxembourg including the one of our university (`uni.lu`) which is probed from internal network. All these domains are presented in figure 5.

5.2 Main metrics

In our experimental evaluation we consider $Init_i$ with $i \in \{SDBF, DNSenum, Fierce\}$, the initial list of discovered domains for each tool and we also define:

$$Init_{overall} = Init_{SDBF} \cup Init_{DNSenum} \cup Init_{Fierce} \tag{8}$$

[1] http://ha.ckers.org/fierce/
[2] http://code.google.com/p/dnsenum/
[3] http://www.wikipedia.org



(a) Horizontal exploration (b) Vertical exploration

Fig. 4. Vertical and horizontal depth analysis (average overall domain)

For the evaluation, we present New_i with $i \in \{SDBF, DNSenum, Fierce, overall\}$ the set of new discovered domains thanks to every initial dataset $Init_i$. Assuming $|S|$ as the cardinality of a set S, the improvement is defined as:

$$\%Imp_i = \frac{|New_i|}{|Init_i|}, i \in \{SDBF, DNSenum, Fierce, overall\} \tag{9}$$

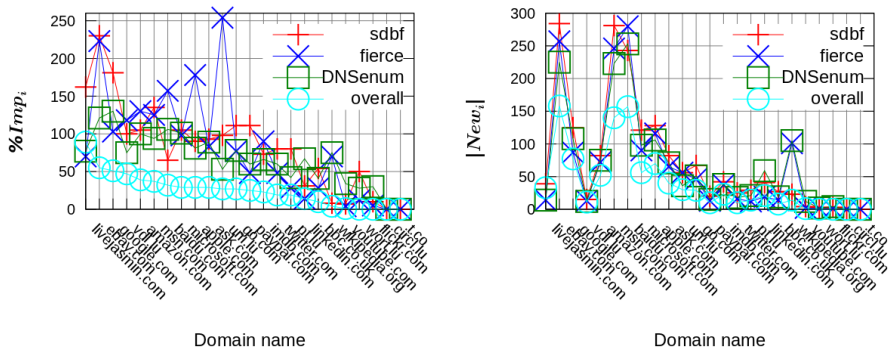
It represents the percentage of new discovered names regarding to the initial dataset. A significant value of $\%Imp_i$ shows that our method is able to find new hostnames which previous methods have not found, even when they are combined.

5.3 Exploration Parameters

Horizontal search: The horizontal search may be configured by adjusting lim_h, which limits the exploration to the top lim_h similar words, as noticed in section 4. On the one hand, we can assume that the more words we have and test, the more hostnames we find. On the other hand, each DNS request is expensive in time and this may lead to the detection of the DNS probe. Figure 4(a) represents the evolution of the hostname discovery regarding lim_h, which varies between 1 and 200. The plotted metric, $ImpH_{i,h}$ represents the proportion of new discovered names when $lim_h = h$ compared to $lim_h - 1$. Assuming $\%Imp_{i,h}$, the value of $\%Imp_i$ when $lim_h = h$, we define:

$$ImpH_{i,h} = \begin{cases} \%Imp_{i,h} & \text{if } h = 1 \\ \%Imp_{i,h} - \%Imp_{i,h-1} & otherwise \end{cases} \tag{10}$$

Figure 4(a) shows that having an exploration limit higher than 40 words does not significantly improve the results. That is why we set lim_h to 40 but, in case a deep domain investigation is required, by increasing it, it can still discover new names, as the curves are still positive. Besides, performances are equivalent, whatever the initial tool is.



(a) Percentage ($\%Imp_i$) of newly discovered hostnames

(b) Number ($|New_i|$) of newly discovered hostnames

Fig. 5. Efficiency of semantic exploration

Vertical search: As our probing method is based on previous discovered hostnames, we can launch it over new hostnames, gathered by the different process iterations. This number of performed probes is called the vertical depth and fixed through lim_v. The process also stops once no new generated names are valid (see section 4). In our case, this leads to a maximal number of 5 iterations. Figure 4(b) represents the ratio of discovered names compared to the maximum ($lim_V = 5$). Between 55 and 80 % of the domain names are found in the first iteration and more than 95 % before the fourth one, so we can reasonably limit the probe to three iterations.

5.4 Gain evaluation

Figure 5 shows the result of our probe, made on 24 domain names using DISCO with the previously tuned parameters. Regarding the individual improvements, in many cases the number of discovered hostnames is doubled ($\%imp > 100$) or even more. For instance with the original dataset from SDBF, the number of names related to domains, as `go.com`, `msn.com` or `google.com`, is increased by more than 100 %, moreover for `ebay.com`, we reach an improvement of more than 200 % for both, SDBF- and Fierce-based intialization. Similar results can be observed for DNSenum and the mean improvement over the 24 domains is between 84% and 102% as shown in Table 2.

Furthermore, this tool provides a real solution to discover new hostnames that existing solutions are unable to find, even if all the three other tools are combined (`overall` in table 2). For instance, a global improvement of 55% for `ebay.com`, 51% for `google.com` or 30% on the overall domains set is observed.

This proves the usefulness and accuracy of semantic exploration as the most common hostnames have already been discovered by one of the initial tools (SDBF, Fierce or DNSenum). From a domain name such as `mars.pt.lu`, `merkur.pt.lu`

	SDBF			Fierce			DNSenum			Overall		
Domains	$\|Init\|$	$\|New\|$	$\%Imp$	$\|Init\|$	$\|New\|$	$\%Imp$	$\|Init\|$	$\|New\|$	$\%Imp$	$\|Init\|$	$\|New\|$	$\%Imp$
livejasmin.com	24	39	162	20	14	70	18	14	77	37	33	89
ebay.com	123	284	230	115	257	223	185	225	121	284	158	55
google.com	69	125	181	84	87	103	83	108	130	149	77	51
vdl.lu	15	15	100	11	13	118	16	12	75	23	11	47
amazon.com	78	82	105	55	72	130	75	75	100	132	52	39
msn.com	207	281	135	196	246	125	236	223	94	372	140	37
baidu.com	369	243	65	178	280	157	238	253	106	478	157	32
microsoft.com	115	121	105	91	90	98	97	98	101	189	56	29
apple.com	141	128	90	65	116	178	130	106	81	241	70	29
ask.com	88	82	93	78	65	83	79	71	89	135	40	29
all domains	2057	1739	84	1520	1558	102	1788	1565	87	3170	954	30

Table 2. Probing results – top 10 and over all domains

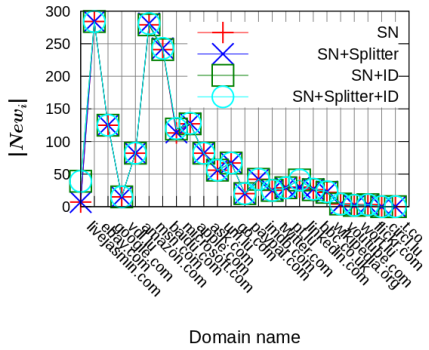


Fig. 6. Efficiency of the different semantic extension when initialized with SDBF

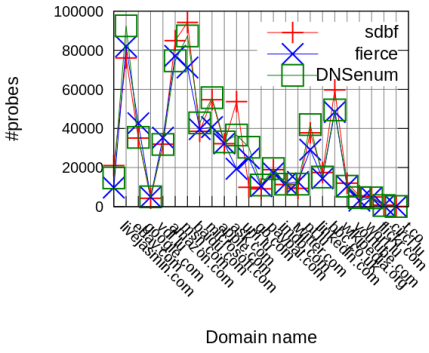
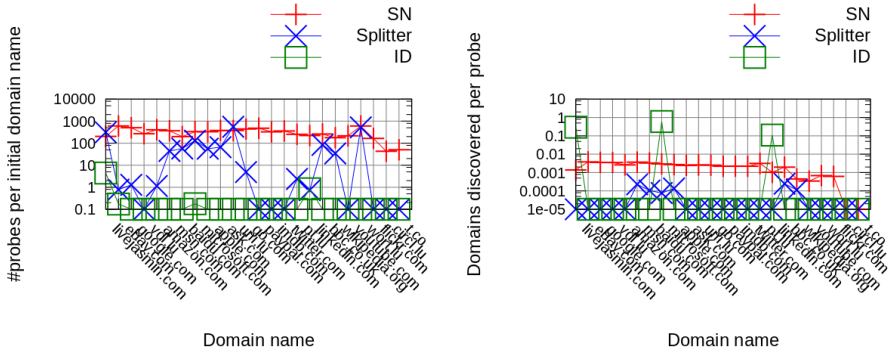


Fig. 7. Number of probes per domain to discover $|New_i|$

and `jupiter.pt.lu` have been found or from `kangaroo.apple.com`, we discover `camel.apple.com`, `porcupine.apple.com` and `piglet.apple.com`. Our first assumption deduced from observations that hostnames are attributed by human and by this, a semantic relation exists between hostnames, proves correct.

5.5 Strategy evaluation

As introduced in section 5.1, different strategies are tested by combing DISCO (SN - *similar names*), the splitter and the incremental modules. Figure 6 shows the efficiency of each startegy initialized with SDBF. We clearly see that *Similar Names* leads to discover the main part of new DNS names, as curves of other strategies mainly coincide with the one from *Similar Names*. The second observation is that *Splitter* provides few signs of improvement to *Similar Names* and *Incremental Discovery* (ID) brings some results, especially for the domain



(a) Number of probes made per domain in the initial dataset

(b) Number of newly discovered hostnames per probe

Fig. 8. Ratio of probes due to each individual module

`livejasmin.com`. In fact through this method, the hostname `news10.livejasmin.com` leads to discover 31 new hosts (`newsX` with $X \in \{1; 9\} \cup \{11; 32\}$).

Therefore, the strategy has to be carefully chosen. For fast probing of many domains, only the DISCO based extension should be used, but if the objective is to probe deeply one domain, all of them have to be combined, since each of them may improve the results.

5.6 Overhead

The overhead is defined as the number of additional DNS requests (*#probes*). As previously mentioned, SDBF and DNSenum require more than 250 000 DNS probes to produce their results. In Figure 7, we can observe that our method always needs to perform less than 100 000 DNS requests, but this discovery is based on a list established by a prior tool. The biggest probes are made for the biggest initial datasets (`ebay.com`, `msn.com`, `baidu.com`) but, half of the domains require less than 20 000 probes. Figure 8(a) shows that the *Similar names* module has a quite steady ratio of probes per initial name (between 200 and 500 requests). The efficiency of this module, as we can see in figure 8(b), is also steady, it discovers around 1 domain name for 200 probes. Other modules perform less requests than the previous one, as we can see in figure 8(a), but figure 8(b) shows that applying *Splitter* is less efficient than *Similar names*, whereas *Incremental discovery* needs to perform very few probes to discover new domains.

These results show that our method is far less expensive than initial ones (at least 4 times for SDBF or DNSenum) for approximatively discovering the same number of domain names (section 5.4). As a basis the *Similar names* module should be used, which provides the steadiest results although, the efficiency of the other tools is dependent of the targeted domain.

6 Related work

In DNS research, major works deal with the detection of DNS attacks as for example, fast-flux, spamming, anomalies in DNS traces,... and present mots various defensive measures for these threats. Statistical evaluation is used in [2], respectively whitelists and classifiers are referred to, to detect anomalous patterns in RR data for rervealing poisoning attacks. The authors in [3] describe a large-scale passive DNS tool, where features are used to detect anomalies, as for example euclidean distances between entries to identify changes in the lifetimes of domains, etc. In [18], suspicious flux networks are detected by passively capturing DNS traffic. The data evaluation is based on the Jaccard index, similar to [8]. To classify the services, the authors refer to supervised learning, where the C4.5 algorithm is used to separate malicious flux and benign services. In [19], the authors perform analysis and visualization of DNS traffic in different modes, off-line, near-real-time and real-time by combining aggregation to clustering. In [6], the authors show that regular expressions improve filtering capabilities for malicious domain detection and provide more accurate results than black-lists.

In this paper, a more semantic approach is used to explore domains in the Net. Natural language processing (NLP) techniques emerged in the research areas of forensics and security. In [5], an automatic domain name generator is constructed by combining different NLP techniques, as for example by using a syllable to construct new passwords or usernames. A major difference to this work is, in [5] full words are generated. By using different statistical tools, as Kulback-Leibler divergence or Levenshtein edit distances, domain names related to botnets can be detected [24]. In the same context of generating new passwords is the work presented in [23]. Here, a new approach relying on probabilistic context-free grammar is used to generate rules in order to crack passwords.

7 Conclusion

In this paper, DNS brute forcing tools are enhanced by using semantics, *i.e.* the average improvement is higher than 80%. When combined with SDBF, the tool only needs a passive DNS database and a set of text samples like Wikipedia. Hence, it may easily be applied and it can be continuously reinforced since the previous databases are continuously evolving. Depending on the context, this paper has assessed the benefit of different strategies, as well as the implied overhead. Future work will deal with distributed probing.

References

1. Backtrack linux - penetration testing distribution (accessed on 08/22/11), `www.backtrack-linux.org`
2. Antonakakis, M., Dagon, D., Luo, X., Perdisci, R., Lee, W., Bellmor, J.: A centralized monitoring infrastructure for improving dns security. In: Recent Advances in Intrusion Detection (RAID). Springer Berlin (2010)
3. Bilge, L., Kirda, E., Kruegel, C., Balduzzi, M.: Exposure: Finding malicious domains using passive dns analysis. In: Network and Distributed System Security Symposium - NDSS (2011)

4. Budanitsky, A., Hirst, G.: Evaluating wordnet-based measures of lexical semantic relatedness. Comput. Linguist. 32 (March 2006)
5. Crawford, H., Aycock, J.: Kwyjibo: automatic domain name generation. Software Practice and Experience 38, 1561–1567 (November 2008)
6. Dagon, D., Lee, W.: Global internet monitoring using passive dns. In: Proceedings of the 2009 Cybersecurity Applications & Technology Conference for Homeland Security. pp. 163–168. IEEE Computer Society, Washington, DC, USA (2009)
7. Faltstrom, P., Hoffman, P., Costello, A.: Internationalizing Domain Names in Applications (IDNA). RFC 3490 (Proposed Standard) (Mar 2003), `http://www.ietf.org/rfc/rfc3490.txt`, obsoleted by RFCs 5890, 5891
8. Hao, S., Feamster, N., Pandrangi, R.: An internet wide view into DNS lookup patterns. Tech. rep., School of Computer Science, Georgia Tech (june 2010)
9. Hindle, D.: Noun classification from predicate-argument structures. In: 28th annual meeting on Association for Computational Linguistics - ACL. Association for Computational Linguistics (1990)
10. Kamra, A., Feng, H., Misra, V., Keromytis, A.: The effect of dns delays on worm propagation in an ipv6 internet. In: Proceedings of IEEE Infocom. IEEE, Miami, FL, USA (2005)
11. Kilgarriff, A.: Thesauruses for natural language processing. In: Natural Language Processing and Knowledge Engineering, 2003 (oct 2003)
12. Kolb, P.: Experiments on the difference between semantic similarity and relatedness. In: 17th Nordic Conference of Computational Linguistics NODALIDA. Northern European Association for Language Technology (2009)
13. Lin, D.: Automatic retrieval and clustering of similar words. In: 17th international conference on Computational linguistics - COLING. Association for Computational Linguistics (1998)
14. Manning, C., Schütze, H.: Foundations of Statistical Natural Language Processing. MIT Press, Cambridge, MA, USA (1999)
15. Mockapetris, P.: Rfc 1035: Domain names - implementation and specification
16. Mockapetris, P.: Rfc 1034: Domain names - concepts and facilities (1987)
17. Mockapetris, P., Dunlap, K.: Development of the domain name system. In: Proceedings of the 1988 ACM SIGCOMM. pp. 123–133. IEEE Computer Society, Stanford, CA, USA (1988)
18. Perdisci, R., Corona, I., Dagon, D., Lee, W.: Detecting malicious flux service networks through passive analysis of recursive dns traces. In: Proceedings of ACSAC'09. pp. 311–320 (2009)
19. Plonka, D., Barford, P.: Context-aware clustering of dns query traffic. In: Proceedings of the 8th ACM SIGCOMM conference on Internet measurement. pp. 217–230. IMC '08, ACM, New York, NY, USA (2008)
20. Segaran, T., Hammerbacher, J.: Beautiful Data: The Stories Behind Elegant Data Solutions, chap. 14. O'Reilly Media (2009), `http://norvig.com/ngrams/`
21. Wagner, C., François, J., State, R., Engel, T., Dulaunoy, A., Wagener, G.: Sdbf: Smart dns brute-forcer. In: To appear in IEEE/IFIP Network Operations and Management Symposium - NOMS, Miniconference. IEEE Computer Society (2012)
22. Weimer, F.: Passive DNS replication. In: Conference on Computer Security Incident Handling (2005)
23. Weir, M., Aggarwal, S., Medeiros, B.d., Glodek, B.: Password cracking using probabilistic context-free grammars. In: Symposium on Security and Privacy. IEEE
24. Yadav, S., Reddy, A.K.K., Reddy, A.N., Ranjan, S.: Detecting algorithmically generated malicious domain names. In: Proceedings of the 10th annual conference on Internet measurement. pp. 48–61. IMC '10, ACM, New York, NY, USA (2010)