2013 IEEE Congress on Evolutionary Computation
June 20-23, Cancún, México

Analyzing String Format-Based Classifiers For Botnet Detection: GP and SVM

Fariba Haddadi, A. Nur Zincir-Heywood
Computer Science, Dalhousie University
Halifax, NS, Canada
{haddadi, zincir@cs.dal.ca}

***Abstract*—The domain name system (DNS) is an essential component of Internet. As it is expected to be used by all legitimate users and applications, generally there are less inspections, restrictions and filters on it. Botnets rely on this open component to accomplish their malicious operation. Therefore, to defeat the single point of failure and evade static blacklists and firewalls, they employ DNS-based methods to frequently generate new automatic domain names. Stateful-SBB, which is a form of genetic programming (GP), was previously designed and developed by the authors to detect these automatically generated domain names based on minimum a priori information which was shown efficient. In this paper, we compare Stateful-SBB against the String Subsequence Kernel (SSK) and SSK with Lambda Pruning (SSK-LP), which are based on support vector machines (SVM) and also use string format inputs. Analyzing the domain names that each of the classifiers chooses as a part of their solutions in the classification process, we notice that 50% to 63% of the Stateful-SBBs' frequently selected points on the Pareto-front are also used by SSK and SSK-LP, respectively. By analyzing these common domain names, we identify some of the characteristics of the botnet domain names. Moreover, we introduce a pruned version of the Stateful-SBB that resulted in reducing the solution complexity by 83% with the same high accuracy.**

Keywords—evolutionary computation; genetic programming; botnet domain name detection

I. INTRODUCTION

Exponential growth of the Internet and online activities necessitate the existence of a secure infrastructure to protect users' identity and information. Attackers distribute malicious programs that can turn a computer into a bot which is the short term for robot. When this occurs, a computer can perform automated tasks over the Internet, without the owner's knowledge. Attackers typically use bots to infect large numbers of computers. These computers form a network, or a *botnet*. A typical botnet is based on a client/server architecture where the server is referred as a botmaster and the clients are the bots. In the architecture, the botmaster employs a Command and Control (C&C) server to communicate with the bots. In 2010, Dambala Inc. published a paper on the top 10 active botnets that not only introduced the top active botnets of the year but also reported the highly increasing botnet infection rate by the average growth of 8% per week [1]. McAfee thread reports also confirm that this growth continues into 2013 [2]. Moreover, these reports also emphasize that new botnets arise every year. Hence, with the high reported infection rate and the vast range of illegal activities, botnets are one of the main threats against the cyber security [3].

Various detection mechanisms have been proposed and improved by researchers. In response to these detection approaches, botmasters have upgraded their bots or even have changed their methodology. Therefore, identifying the botnets and detecting them have become very challenging. As a complementary technique, an active continuous botnet monitoring and detection mechanism is required which could potentially learn the new patterns through the monitoring process and adapt to the changes in the botnet evolution.

In recent years, most botnets use Domain Name System (DNS) as a way to evade detection mechanisms. DNS translates the domain names that people use to the IP addresses that machines use to locate computers and services on the Internet. DNS is an essential and important functionality for the Internet to work as we know it. Botnets manage to use DNS in two different ways: (i) using DNS packets as the carrier of information and commands between the botnet computers; (ii) locating the C&C server by querying a list of domain names supplied at the time of infection or after. Typically, the list of domain names is algorithmically generated for this purpose. This list of domain names is large enough that it cannot be blacklisted manually or at the firewall level. Given that the algorithmically generated domain names exhibit structural and syntactical anomalies compared to regular domain names, it should be possible to detect them by monitoring high volume access to unusually structured domain names. Thus, most of the existing works in this field employ network DNS traffic behaviour analysis [4] [5] [6] and some works combine such an approach with domain name lexical analysis to improve the performance [7] [8] [9] [10]. However, domain name lexical analysis requires attributes (features) to be defined a priori to work accurately. Yet, the botnets keep changing the architectures, C&C channels and the domain name generation algorithms they use frequently [11] [12] [13]. So it is difficult to keep up with such changes, when the detection systems still use a priori knowledge on botnet architectures or domain name

generation algorithms that belong to the earlier version(s). In our previous work [14], we have designed and developed a GP based system, namely Stateful-SBB, to detect maliciously generated domain names. Stateful-SBB can work with the string format variable length domain name without requiring any a priori known features [14]. We have shown that it performs comparable to other well-known classification methods with the privilege of not requiring any a priori knowledge [14]. Now in this work, we aim to evaluate the Stateful-SBBs' solutions utilizing a much larger dataset (more than seven thousands domain names) to test its generalization capabilities as well as comparing it against other classifiers that can work with string format inputs. To this end, we compare it against String Subsequence Kernel (SSK) as well as SSK with Lambda Pruning (SSK-LP) and analyze the domain names that each chooses as a part of the solution.

The remainder of the paper is organized as follows: Section 2 describes botnets and botnet detection systems. Section 3 discusses the protocols and learning models employed in this work. Evaluation and results are provided in Section 4 and conclusions are drawn and the future work is discussed in Section 5.

II. BACKGROUND

A. Botnets

A bot program is a self-propagating malware that is designed to infect a vulnerable host through exploitation techniques. As discussed earlier, a network of these compromised computers, bots, is called a botnet that is under the remote control of a botmaster via the C&C server. Since DNS is an essential component for the Internet to function, security solutions would not filter out the DNS traffic. Thus, DNS became one of the most powerful protocols to be utilized in malicious activities such as botnets. DNS is used in botnet architectures to provide robustness and mobility as well as to avoid the single point of failure in two categories: (i) Rallying and (ii) Communication [15]. In Rallying, DNS is used to help the bots to locate their C&C server. In Communication, DNS is used as the information carrier, i.e. communication protocol, in the botnet operation. To the best of our knowledge, there are not so many active botnets in the second category since the first fully DNS-based botnet came into existence in 2011 [15]. Having said this, Dietrich et al. claimed to present the first paper that documented DNS-based botnet C&C traffic [16]. Conversely, there are many botnets that use DNS for Rallying purposes (such as Conficker, Kraken, Bobax, Torpig, etc.). These botnets use domain and IP fluxing to be more robust and to avoid being detected. In this context, domain or IP fluxing refers to the technique that a malicious user can use to prevent identification of his/her key server's IP address. By abusing the way the DNS works, a botnet can be created with nodes that can join and drop off the network faster than the security systems can detect them. As is mentioned in different security reports, some of the botnets of this type are in the list of top destructive botnets with high infection rates. For example, Conficker-A was in the top 10 botnets list of Damballa in 2009 [17], Conficker-C was listed in Damballa top 10 botnets of 2010 [1] and Bobax was listed in McAfee thread reports of 2011 and 2012 [18] [19] .

B. Botnet Detection

Botnets employ different protocols (i.e. IRC or HTTP), topologies (i.e. centralized or distributed) and techniques (i.e. fluxing). Hence, different detection mechanisms have been proposed focusing on different aspects. For instance, Strayer et al. proposed a botnet protocol-independent detection approach based on the network flow characteristics [20] while Perdisci et al. developed a detection method for HTTP-based botnet C&C detection [21]. To identify DNS-based botnets, which use DNS for either locating their C&C server or as the information carrier, many works employ only network traffic analysis and/or use domain names textual characteristics

Holz et al. developed a fast-flux service networks (FFSN) detection system based on the recursive DNS traffic analysis [4]. Several malicious fast-flux features such as short Time-to-Live (TTL) and unique Autonomous System Numbers (ASNs) were identified and accordingly, metrics such as Fluxiness and Flux-Score were introduced and measured. Perdisci et al. implemented a FFSN detection system through passively analyzing the recursive DNS traces [5]. The same features, Holz et al. introduced, were identified. Passively collecting the recursive DNS traces from multiple large networks and clustering based on the related domain names, they built a system to identify the fast-flux domain names. Manasrab et al. proposed a botnet detection mechanism based on the fact that botnets appeared as a group of hosts periodically [6]. Monitoring and capturing the DNS traffic at different time intervals and using the Jaccard index to measure the similarity, they developed a system to detect botnets.

There are several other approaches that use domain name lexical analysis along with traffic analysis which are more relevant to our work. Stalmans et al. developed a system to detect fast-flux domain names using DNS queries [7]. Analyzing the DNS query responses, two groups of features were extracted to identify legitimate and malicious queries: DNS features (i.e. unique ASNs and TTL) and Textual features (i.e. alpha numeric characters frequency distributions). Given the extracted features, they employed C5.0 and Bayesian classifiers to identify fast-flux queries. Yadav et al. proposed a methodology to detect malicious algorithmically generated domain names, addressing the domain fluxing mechanism [8]. To this end, they used several methods and features to group the DNS queries. Then for each group, metrics that characterized the distribution of alphanumeric characters- such as Kullback-Leibler divergence and Levenshtein edit distance- were computed and used to identify domain names with structural anomalies. Ma et al. employed supervised learning techniques (i.e. Naive-Bayes, SVM) to detect malicious web sites from suspicious URLs [9]. To characterize the URL's, two categories of features are used: lexical features (the length of a domain name, the number of dots, etc.) and host-based features (IP address properties, WHOIS properties, etc.). Antonakakis et al. presented a dynamic reputation system, Notos [10]. Using DNS query information as well as zone and network features of domain names, Notos builds models of legitimate and malicious domain names. Then, using these models, reputation scores are assigned to label them as malicious or legitimate. Haddadi et al. designed and developed a botnet detection system using evolutionary technique, called Stateful-SBB [14].

The proposed system only uses the raw string format domain names and does not employ any traffic analysis. In addition to automatically generating signatures, Stateful-SBB identifies the set of attributes to be used in detection automatically without any a priori knowledge.

III. METHODOLOGY

In this work, we aim to evaluate the robustness of different string format based classifiers to detect maliciously generated domain names with minimum a priori information. Thus, in this section we will summarize the classifiers employed for this purpose.

To be more flexible, to act more intelligently, to enhance their functionality and to avoid detection systems, botnets tend to use automatic algorithms in several stages of their malicious lifecycle. For example, they use automatic exploitation techniques instead of fixed vulnerability checklists to find security breaches in the computers (victims). They also employ fluxing techniques, using automatic domain generating algorithms (DGAs), to hide the C&C servers. Thus, detection systems also require automatic and intelligent mechanisms to be able to cope with the advanced techniques botnets employ.

As discussed earlier, DNS as the ubiquitous component of Internet has been used in Botnets' topology in several ways [15]. In this category of botnets, some used domain fluxing techniques along with DGAs. To deal with this type of botnets, in our previous work, we developed the Stateful-SBB to differentiate malicious automatically generated domain names (indicating botnets activity) from legitimate ones, which are used in many well-known applications/services such as Google and Facebook. Going a step further, in this work, we gather a new larger data set of domain fluxing botnets' data to test the generalization capabilities of our proposed system. Moreover, given that the final goal of botnet detection systems are differentiating the malicious domain names from the legitimate ones, in this work we intend to adjust and test the proposed Stateful-SBB in order to differentiate the humanly structured legitimate domain names (i.e. domain names listed in Alexa) from the malicious automatically generated domain names.

To the best of our knowledge, all detection systems in this field analyze DNS network traffic behaviour via classifiers with pre-defined feature sets. However, our system, Stateful-SBB, only uses string format data, i.e. the raw domain name strings (no a priori information). To compare Stateful-SBB against a classifier that can also work on string type data, we choose SVM with string kernel in [22]. In addition, given that the C4.5 classifier was the best performer to detect malicious automatically generated domain names from legitimate ones using a pre-defined feature set , i.e. a priori information, [14], it is also used in the evaluations here. We believe this helps us to understand how far we can push a minimum or no a priori information approach as well as understanding how well these approaches (a priori vs. no a priori) generalize to different data sets gathered from different sources, i.e. robustness.

A. SVM With String Kernel

The Support Vector Machine is a binary learning machine that can be used for classification and rule regression [23]. The main idea of this classification algorithm is to build a hyperplane that optimally separates the samples of data into two categories with maximal margin. The classifier can easily be extended to K-class classification by constructing k two class classifiers.

Fig. 1 illustrates the training process of an SVM, where training samples are a set of $\{x_i, y_i\}$ in which x_i is an N-dimensional feature vector and y_i is the class label (for example $y \in \{-1,+1\}$), C controls the misclassification behavior/penalty and α controls the shape of the separating hyperplane. The data points x_i for which $\alpha_i > 0$ are the points on the margin (or within the margin while using soft-margin SVM). These data points are called support vectors which are shown with cycles in fig. 2. Finally, the hyperplane can be represented as (1):

$$w = \sum_i \propto_i y_i x_i \qquad (1)$$

In order to use an SVM to solve a classification problem on a non-linearly separable data, a non-linear mapping of an input data into a high dimensional feature space is required. Then, an optimal hyperplane for separating the high dimensional features of input data can be constructed, which maximizes the separation margin. Finally, a linear mapping from the feature space to the output space is required. Generally, explicitly mapping the non-linearly separable data into a high dimension feature space has a very high computational cost. Therefore, To handle the problem, kernel functions are utilized to implicitly map the data points to the feature space. Mathematically, for any mapping $\phi : D \to F$ the function $K : K(x_1, x_2) = (\phi(x_1), \phi(x_2))$ is a kernel function where $(.,.)$ denotes the dot product. Several kernels have been introduced so far such as Polynomial, Gaussian and Hyperbolic tangent.

Lodhi et al. proposed a string kernel, called SSK (String Subsequence Kernel), for the purpose of text classification [22]. The idea is to define the dot product of two text data points by means of their substrings. The more substrings they have in common, the more similar they are. Note that the substrings do not need to be contiguous. However, the subsequences are weighted by an exponentially decaying factor of their length, emphasizing on the substrings that are close to contiguous. In other words, the contiguity degree of a substring determines its' effectiveness (weight) in the comparison.

The main parameters of SSK are n (subsequence length), and λ (decay factor). The kernel implicitly maps the input string data into a feature space F for every subsequence u of n characters using the following formula, (2).

$$\phi_u(s) = \sum_{i:\, u=s[i]} \lambda^{l(i)} \qquad (2)$$

$\phi_u(s)$ is the overall result for all the occurrences of u in a string s. λ^l is the value of each occurrence of u where l denotes

Algorithm-1

Input: X and y loaded with training labeled data,
α <- 0 or α <- partially trained SVM
1: C <- Some value
2: **repeat**
3: **for all** $\{x_i, y_i\}$, $\{x_j, y_j\}$ **do**
4: Optimize α_i and α_j
5: **end for**
6: **until** no changes in α or other resource constraint criteria met
Ensure: Retain only support vectors ($\alpha_i > 0$)

Fig. 1. SVM training algorithm [24]

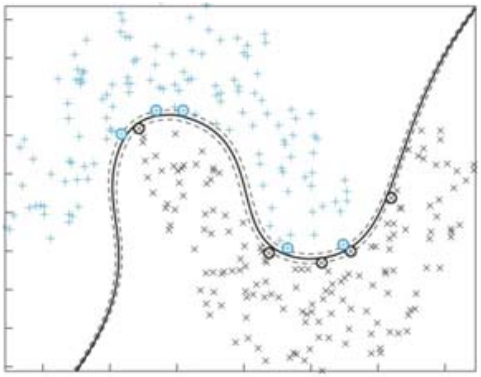

Fig. 2. SVM hyperplane and support vectors [25]

the length of that subsequence in *s* (length of *u* plus interior gaps of the occurrence). Kernel function result for two strings *s* and *t* is the dot product of their features mapping, (3).

$$K_n(s,t) = \sum_{u\in\Sigma^n}\left(\phi_u(s).\phi_u(t)\right) = \sum_{u\in\Sigma^n}\lambda^{l(i)}\sum_{i:u=s[i]}\sum_{j:u=t[j]}\lambda^{l(j)}$$
$$= \sum_{u\in\Sigma^n}\sum_{i:u=s[i]}\sum_{j:u=t[j]}\lambda^{l(i)+l(j)} \quad (3)$$

Given that even for a small substring size (i.e. four) and an average sized text, direct computation of feature space is impractical. Thus, SSK utilizes an efficient recursive formulation using dynamic programming techniques with the complexity of *O(n|s||t|)*.

B. SSK-LP

Seewald et al. introduced SSK-LP (SSK-Lambda Pruning) to decrease SSK computational time and memory consumption along with a little loss in accuracy. In SSL-LP, the recursion is stopped as soon as it gets to an acceptable result in the current branch. This approach decreases the computational effort required for the SKK. A detailed explanation of the algorithm can be found in [26].

C. Stateful-SBB

The SBB is a genetic programming-based learning algorithm [27]. It coevolves three populations: A point population, a team population and a learner population. The learner population represents a set of symbionts (learners), which associate a GP-bidding behaviour with an action. The team population comprises a set of learners and finally the point population denotes a subset of training data exemplars. Although all of the teams' learner programs are executed while evaluating a team on the points, only the learner with the highest bid suggest its action. The bidding procedure employs linear GP. Each program consists of a sequence of instructions representing one- and two-oprand operations ({*cos, exp, log, +, -, *, %*}) applied to inputs and a set of null-initialized registers.

The point and the team population interaction follows a Pareto-based competitive coevolution. In this concept, if an individual is not dominated by any other individual, it is set to be a part of Pareto-front. This relation is used by SBB training algorithm to determine the points and the teams, which survive to the next generation. At each generation, P_{gap} new points are generated by sampling the training data and M_{gap} new teams are generated through variation operators (add, delete, swap and mutate) applied to the existing teams, while learners in both parent teams are copied into the offspring. Following the fitness evaluation of all teams against all points, P_{size}-P_{gap} points and M_{size}-M_{gap} teams are opted to appear in the next generation using a Pareto-based selection. Pareto competitive coevolution ranks the teams' performance, the Pareto non-dominated teams with the highest ranks are selected. Likewise, the non-dominated points are also preserved. Meanwhile, if a point/team ranking is required in these non-dominated subsets, a form of competitive fitness sharing is employed in order to bias in favour of the points/teams that exhibit non-overlapping behaviour.

As many other classifiers, SBB requires data to be represented using a fixed length feature set. This means SBB learners bid on the points represented by the whole feature set. Therefore, the variable length string format domain names cannot be used neither in SBB, nor in any other standard classifier such as C4.5, SVM or Naive-Bayes. Instead, a useful set of features need to be determined, to properly represent the domain name characteristics. Indeed, this is a challenging task and requires a priori knowledge. Once the feature set is determined to represent the domain names, then a feature extraction procedure should be applied to the data set prepare it for the classifiers. To avoid these data processing steps, we designed and developed a new version of SBB, which keeps the state information for each exemplar; hence, we called it the Stateful-SBB [14]. Before describing the new model, there are three important characteristics of learners in association with the bidding mechanism that should be mentioned. First, each learner bids on the point separately but only the learner action with the highest bid is returned as the team action. Second, using the data set with a fixed number of features for the exemplars, the learners bid on each point based on the whole feature set. Finally, each learner resets its registers before bidding on the next point.

To deal with the variable length domain names (in their text format), we changed the original SBB interface to bid based on each character of a domain name each time. Features for this model are the ASCII codes of the domain names' characters. A team receives a complete domain name but passes it character by character to its team of learners. Each learner then bids per character as opposed to per exemplar. The learners' action that outbids the others is assigned as the team output for that specific character. Domain name characters are related to each other and are not independent. To achieve the correlation of characters reflected in the bidding process, learners reset their registers only at the beginning of each specific domain name, not for every bid process on every character in a domain. At the end of each domain name (when all the learners bid on the entire domain name characters), a team will output a sequence of the best learners' actions. Finally, the team will decide on its final action for that specific domain name. Different policies can be used for the final action selection of a team. Based on our previous results [14], in this work, we employ the "Last-best" method for the action selection procedure.

D. Pruned Stateful-SBB

In Stateful-SBB, all the generated teams in the learning procedure are evaluated on the training data set and the one

with the best performance is selected as the final solution. The solution team is a combination of a set of learners with their corresponding GP instructions. Since the maximum program size is 48 in parameters, each learner in the solution can at most have 48 instructions including non-effective codes, called introns. Given that introns were found to count for between 60% to 90% of instructions in linear GP, intron removal would lead to major improvement [28]. Thus, in this work, we employ intron removal to reduce the complexity of Stateful-SBB.

E. C4.5

C4.5 is a decision tree, which is a non-parametric supervised learning method, based classification algorithm first developed by Quinlan [29]. Given that C4.5 is an extension to the ID3 algorithm, it is designed to address the issues that are not covered by the former algorithm such as removing the restriction that features must be categorical, converting the output of ID3 algorithm, trained tree, into an if-then rule set, measuring rules' accuracy to determine their order of being applied and finally pruning [23].

C4.5 constructs decision trees based on a training data set, applying the Information Entropy concept. The algorithm employs a normalized information gain criterion to select attributes from a given set of attributes to determine the splitting point. In other words, the feature with the highest information gain value is chosen as the splitting point. A decision node is then generated based on this selected point. The training process then recursively continues on the corresponding sub-lists obtained until all of the data samples associated to the leaf nodes are of the same class or the classifier runs out of training samples. To this end, purity of all branches is measured. A branch is pure only if all of the associated samples are of the same class. If so, the splitting process stops. If the split is not pure, the instances should be split to decrease the impurity. A detailed description of the learning algorithm can be found in [23].

IV. EVALUATION AND RESULTS

A. Data Set

The malicious data employed in this work is collected from two publically available botnet C&C domain name lists, Conficker [30] [31] and Kraken [32] [33] [34]. Given that Conficker was one of the top 10 active botnets in 2009 and 2010 and Kraken has been active for the last few years, these botnets have the most publically available domain name for filtering out traffic as "blacklists". Hence, we employ these lists to form the malicious exemplars in our data set. Conficker-A generates 250 domain names every 3 hours. Conficker-B is a rewrite of Conficker-A with some differences which generates 250 domain names every 2 hours with different seeds and additional TDLs (Top Level Domain Names) [11]. To detect Conficker operations, some organizations attempted to pre-register these daily domain names lists to prevent the bots to connect to the C&C servers and get the updates or commands [35]. To avoid this defensive procedure, Conficker-C creates 50000 domain names per day, making the pre-registration almost impossible due to its high cost [12]. On the other hand, the botnet called Kraken uses a more sophisticated DGA called the dictionary based generator to generate new domain names. The generator employs a complicated random word generator and creates words that look like English words and then combines them with randomly selected parts of common English words/phrases (noun, verb, adjective) as suffixes [36]. For some detection mechanisms that detect the automatically generated domain names based on the fact that these domains do not follow common and pronounceable phrase rules in English, the detection procedure gets much more complicated.

In addition to Conficker and Kraken, some of the most frequently requested domain names from the Alexa lists are used to represent legitimate domain names [37]. Alexa Internet Inc. ranks websites based on page views and unique site users and accordingly, publishes the list of most common websites. Given that even the Alexa lists may have malicious domain names [38], we manually extracted 500 benign domain names from the Alexa list for our dataset employed in this work[1].

Each domain name has three components: (i) TLD, (ii) core domain, and (iii) sub-domain. For example, in "mail.google.com", "com" is the TLD, "google" is the "core" domain, and "mail" is the sub-domain. We applied two filters on the Conficker and the Kraken data sets to obtain a more balanced data set: (i) Domain names that have at least three parts (Sub-domain, core and TDL) and (ii) Domain names that have valid IP addresses assigned to them. Table I. details the data set employed in this work. We divided the dataset into two parts (training and testing) based on: (i) An almost 30-70% breakdown for testing and training, respectively; and (ii) keeping enough samples of each class in both of the datasets.

B. Performance Criteria

1) Score: Typically, classifiers are evaluated using accuracy or classification rate as the fraction of all the correctly labeled instances. However, given an unbalanced data set or a multi-class data set, these metrics can be misleading. In this regard, we employ a classwise detection rate, (4), [39].

$$DET_c = \frac{TP_c}{FN_c + TP_c} \quad (4)$$

DET_c is the class c detection rate and TP_c and FN_c are the True-Positive and False-Negative counts for class c. They also present Score criteria to summarize the classwise detection rates of a classifier over all classes, (5).

$$Score = \frac{1}{|C|}\sum_{c \in C} DET_c \quad (5)$$

2) Complexity: Definition of complexity often depends on the concept of the "system". Speaking of classifiers, complexity can be measured on different criteria such as memory consumption, time or solution. In this work, two complexity criteria is utilized. Firstly, computation time, which is a typical scale for learning algorithms during training procedure denoted as training time. After a classifier is trained,

TABLE I. DATA SETS EMPLOYED IN THIS WORK

Data Type	Training	Testing
Kraken (Class 0)	4017	1722
Conficker (Class 1)	654	280
Alexa (Class 2)	350	150

1 http://web.cs.dal.ca/~haddadi/Alexa-list.pdf

the trained model is presented as the solution to be used for testing purposes. Given that presenting a better solution to a problem is important, we define the solution complexity as our second criteria. A direct comparison between solutions of different approaches is impractical since the underlying units of measurement are different. Therefore, different elements are: For C4.5, the tree size; for SSK and SSK-LP, the number of support vectors, and finally for Stateful-SBB, the program size of the solution team.

C. Results

As discussed earlier, C4.5 and SSK classifiers will be compared against Stateful-SBB on our data set. Among these three classifiers, C4.5 requires the data set to be represented via a priori known feature set for training and testing purposes. Therefore, feature extraction procedure was applied to the training and testing data set using the features introduced in [14]. On the other hand, SSK, SSK-LP and Stateful-SBB can use raw domain names in string format without requiring any a priori known feature set. For SKK, SKK-LP and Stateful-SBB, each testing and training data exemplars have two attributes: Domain name (string format) and Class label.

We run C4.5, SSK, SSK-LP, Stateful-SBB and pruned Stateful-SBB several times changing different parameters (C parameter and pruning option for C4.5, Lambda and subsequence length for SSK and SSK-LP and finally maximum team size and seed number for Stateful-SBB). Table II. presents the best result of each classifier on our data sets. Given that our goal is to have a low complexity with a high Score, which are most of the times conflicting, the decision of the best run need to be taken in the presence of trade-offs between these two criteria.

Not to cause any changes to the individuals in the learner population during evolution, we removed the non-effective instructions after the training. Therefore, the training time of Stateful-SBB and Pruned Stateful-SBB are the same, as is indicated in Table II. Given that we only removed the introns from the learners' instruction set, TP and FP rates are also not changed, hence the Score. Since training is a onetime process but the solution will be used several times, our goal of pruning Stateful-SBB was to reduce the solution complexity which will speed up the botnet malicious domain name detection process in real-time. The results indicate that using the pruned version of Stateful-SBB reduces the complexity by 83% which is a significant improvement. On the other hand, the SSK-LP reduces the complexity of SSK by 27%. These results suggest that the Pruned Stateful-SBB is a very promising classifier as an automatically built malicious domain name detector, which can achieve very high accuracy with no a priori known feature set. On the other hand, C4.5 has a very low training time and solution complexity but it requires a priori known set of features to be able to achieve a high accuracy. Given that the recent botnets change their DGAs almost on the fly, we believe that classifiers working with a priori known feature sets may not be able to cope with this in practice.

Furthermore, comparing the SSK-LP with Stateful-SBB, we go a step further and analyze the domain names that each of them selected to use as a base of classification. Point population Pareto-front in Stateful-SBB is equivalent to the support vectors in the SSK and SSK-LP, called critical points from now on. Since Stateful-SBB point selection is heuristic but SSK, or SVM in general, tests all of the samples, we compare the Pareto-front points used in 20 runs of Stateful-SBB and the support vectors of the best runs of SSK and SSK-LP. Table III. shows the critical point counts. Given that SSK and SSK-LP are binary classifiers, to build a 3-class classifier, in our case, a combination of three binary classifiers are utilized, hence some of the support vectors are used more than once. Stateful-SBB, SSK and SSK-LP used 868, 844 and 669 critical data points, respectively. These critical data points represent the most important domain names that aid in the pattern discovery (classification) process.

Fig 3. shows the training data set where each point represent a specific domain name. As it is indicated in the figure, Kraken data points comprise three categories: (i) "$^{+}$.dynserv.com"; (ii) "$^{+}$.mooo.com", and (iii) "$^{+}$.yi.org"[2]. In the first two categories, Stateful-SBB and SSK have selected almost the same number of points as part of their solution while only 0.6% of these points are chosen by both of the classifiers. Analyzing these points showed that the sub-domain section of the points have relatively the same structure in case of the characters, distribution and combination. This means, many of the data points in "$^{+}$.dynserv.com" and "$^{+}$.mooo.com" categories would play the same role if chosen as the critical points. On the other hand, "$^{+}$.yi.org" data points are seemed not to follow a specific structure. Almost all of the data points in this category are used in Stateful-SBB or SSK where 50% of them are used in both of the classifiers. No specific feature could be identified analyzing these 50% of the domain names.

Fig 3. indicates that Conficker data points can be divided into three main categories: (i) "$^{+}$.co*"; (ii) "$^{+}$.com*", and (iii) "$^{+}$.ws". Stateful-SBB and SSK employ 47% and 35% of the data points of the first two categories, respectively, where 20% of them are common between the two classifiers. Unfortunately, not any explicit feature could be identified while analyzing these common data points. On the other hand, there are a few exemplars in Conficker data set that belongs to "$^{+}$.ws" category. Given that these data points are long in length

TABLE II. RESULTS

Classifier	Training time	Score	Kraken		Conficker		Alexa		Solution Complexity
			TP	FP	TP	FP	TP	FP	
Stateful-SBB	2227.64	0.983	0.984	0.004	0.993	0.001	0.973	0.015	674
Pruned Stateful-SBB	2227.64	0.983	0.984	0.004	0.993	0.001	0.973	0.015	116
SSK	431.53	0.996	1	0	0.989	0	1	0.001	1094
SSK-LP	166.2	0.994	1	0	0.989	0.001	0.993	0.001	805
C4.5	0.06	0.937	1	0.014	0.964	0.01	0.847	0.004	19

[2] '$^{+}$' means more than one and '*' means zero to many characters

TABLE III. CRITICAL POINTS COUNT

Classifier	Total Num. of Critical Points	Total Num.of Points (no duplicates)	Range of Frequency
Stateful-SBB	2380	868	1-20 times
SSK	1049	844	1-2 times
SSK-LP	805	669	1-2 times

with no observable structure and few in number, all of them are used by both of the classifiers. Finally, in Alexa data points, two categories are evident, Fig. 3: (i) "$^{+}$.co/com.*, and (ii) others. Stateful-SBB and SSK used 26% and 20% of the domain names in each category, respectively. However, only 20% are employed in both of the classifiers. Analyzing these 20% of the data points, we understand that they are the longest domain names of the Alexa list with the average length of 20. Given that the average domain name length of Conficker, Kraken and Alexa are 18.9, 16.7 and 10.9, respectively, these 20% data points are more close to the botnet data points. This could be one of the reasons that both of the Stateful-SBB and SSK classifiers chose these exemplars as the critical points.

Overall, almost 30% of the points of Stateful-SBB overlap with the data points selected by SSK and SSK-LP. Of these 868 domain names (critical data points), Stateful-SBB used 155 of them more than three times of which 63% and 51% are also used in SSK and SSK-LP, respectively, as shown in Fig. 4. This means 50% to 60% of the Stateful-SBB highly frequent domain names are also selected in SVM string kernel classifiers. Analyzing the common critical points between Stateful-SBB (with frequencies of 4 and more), SSK and SSK-LP: In class Kraken, 50% belong to the "yi.org" category; in class Conficker, 57% belong to the ".ws" category; and finally in class Alexa, 91% of the domains belong to ".co/com" category. Further analysis on these frequently used domain names may aid us to identify automatically generated malicious domain name characteristics which lead us to design a better detection system to recognize botnet behaviour.

As discussed in section 3, Kraken botnet uses a dictionary-based generator to create the domain names that are more similar to human generated domain names. However, we did not observe this in our results and analysis on our data set. On the contrary, in our experiments, most of the misclassifications of Stateful-SBB, SSK and SSK-LP classifiers, are between class-1 (Conficker) and class-2 (Alexa). This indicates that Conficker domain names are more realistic and more similar to Alexa legitimate domain names. Moreover, our observation of Kraken and Conficker domain name categorization also indicate that while Conficker is using ".com" and ".co" as a part of its domain name, which is more like Alexa domain names, Kraken is using predefined "dynserv.com" and "mooo.co" and only generates the sub-domain section algorithmically. Indeed, more data sets need to be analyzed to confirm our observations.

V. CONCLUSION

Not only legitimate users utilize DNS to communicate in the network but also botnets employ DNS-based techniques to avoid 'hardcoding' the address of the C&C server and therefore, to avoid being detected at the firewall level. To this end, domain-fluxing technique is utilized to generate a large list of

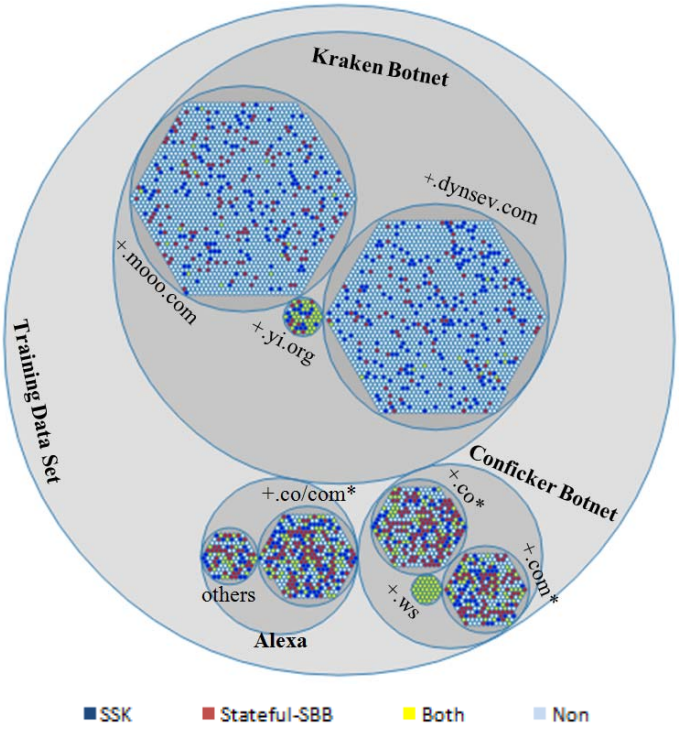

Fig. 3. Training data points

algorithmically generated domain names. The domain names of the list will be then queried by the victim host when required to locate the C&C server. Fortunately, the structure of the domain names is different from legitimate domain names and querying these domains leave abnormal footprints in the network DNS traffic. Therefore, these anomalies can be employed to detect the botnets that use this method.

Previously, Stateful-SBB was presented by the authors to differentiate malicious automatically generated domain names from legitimate ones. The proposed system was proved to be efficient in case of accuracy while having the vantage of not requiring any pre-defined feature set (no a priori knowledge). In this work, a pruned version of Stateful-SBB is proposed that reduces the solution complexity by 83%. The Stateful-SBB and Pruned Stateful-SBB were then evaluated on a larger data set of malicious algorithmically generated domain names as well as Alexa Legitimate domains and compared against SSK and SSK-LP (SSK with Lambda pruning) which can also use string format input. The results indicate that Stateful-SBB performs comparable to SSK and SSK-LP based on classwise detection rate, called Score. On the other hand, it is shown that the Pruned Stateful-SBB is more effective in terms of solution complexity. In other words, Pruned Stateful-SBB has a very competitive detection capability with less rules. This in return speeds up the real-time detection process. Further analysis was done on the Stateful-SBB point Pareto-fronts and their equivalent, SSKs' support vectors. The results show that more that 50% of highly used point Pareto-fronts of SBB were also used in SSK and SSK-LP. The analysis of the common points also reveals some textual features of the botnet and Alexa (non-botnet) domain names. Future work will include more detailed textual analysis of the botnets' domain names.

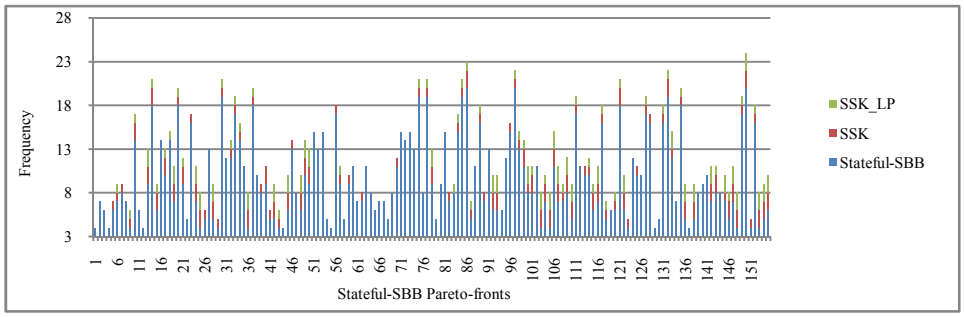

Fig. 4. Stateful-SBB most frequent critical points

ACKNOWLEDGMENT

This research is supported by the Natural Science and Engineering Research Council of Canada (NSERC) grant, and is conducted as part of the Dalhousie NIMS Lab at http://projects.cs.dal.ca/projectx/

REFERENCES

[1] Damballa, "Top 10 Botnet Threats," Damballa Inc., 2010.

[2] McAfee Thread Reports. [Online]. http://www.mcafee.com/apps/view-all/publications.aspx

[3] M. Feily and A. Shahrestani, "A Survey of Botnet and Botnet Detection," in Emerging Security Information, Systems and Technologies, 2009.

[4] Ch. Holz, Ch. Gorecki, K. Rieck, and F.C. Freiling, "Measuring and Detecting Fast-Flux Service Networks," in NDSS, 2008.

[5] R. Perdisci, I. Corona, D. Dagon, and W. Lee, "Detecting Malicious Flux Service Networks through Passive Analysis of Recursive DNS Traces," in ACSAC, 2009.

[6] A. Manasrab, A. Hasan, O.A. Abouabdalla, and S. Ramadass, "Detecting Botnet Activities Based on Abnormal DNS traffic," in *IJCSIC*, vol. 6, no. 1, pp. 97-104, 2009.

[7] E. Stalmans and B. Irwin, "A Framework for DNS Based Detection and Mitigation of Malware Infections on a Network," in ISSA, 2011.

[8] S. Yadav, A.K.K. Reddy, A.L. Narasimha Reddy, and S. Ranjan, "Detecting Algorithmically Generated Domain-Flux Attacks With DNS Traffic Analysis," in *IEEE/ACM Transaction on Networking*, 2012.

[9] J. Ma, L.K. Saul, S. Savage, and G. Voelker, "Beyond blacklists: Learning to detect malicious Web sites from suspicious URLs," in ACM KDD, 2009.

[10] M. Antonakakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster "Building a Dynamic Reputation System for DNS," in USENIX Security, 2010.

[11] Ph. Porras, H. Saidi, and V. Yegneswaran, "An Analysis Of Conficker's Logic And Rendezvous Points", 2009. [Online] http://mtc.sri.com/conficker

[12] Ph. Porras, H. Saidi, and V. Yegneswaran, "Conficker C Analysis", 2009. [Online] http://mtc.sri.com/Conficker/addendumC/index.html

[13] "Kelihos Back In Town Using Fast Flux", 2012. [Online] http://www.abuse.ch/?p=3658

[14] F. Haddadi, H.G. Kayacik, A.N. Zincir-Heywood, and M.I. Heywood, "Malicious automatically Generated Domain Name Detection Using Stateful-SBB," in EvoApplication, 2012, in press.

[15] "The Role of DNS in Botnet Command & Control," Open DNS Inc., Whitepaper, 2012.

[16] Ch.J. Dietrich, Ch. Rossow and F.C. Freiling, "On Botnets that use DNS for Command and Control," in EC2ND, 2011.

[17] O. Gunter: Damballa, "Top-10 Botnet Outbreaks in 2009", 2009. [Online]. http://blog.damballa.com/?p=569

[18] McAfee Labs TM, "McAfee Threats Report: First Quarter 2011," McAfee, 2011.

[19] McAfee Labs TM, "McAfee Threats Report: First Quarter 2012," McAfee, 2012.

[20] W. Timothy Stayer, D. Lapsely, R. Walsh, and C. Livadas, "Botnet Detection Based on Network Behavior," in *Botnet Detetcion*.: Springer, pp. 1-24, 2008.

[21] R. Perdisci, W. Lee, and N. Feamster, "Behavioral Clustering of HTTP-based Malware and Signature Generation using Malicious Network Traces," in NSDI, 2010.

[22] H. Lodhi, C. Saunders, J. Shawe-Tylor, Nello Cristianini, and Ch.J.C.H. Watkins, "Text Classification using String Kernels," In *Journal of Machine Learning Research*, vol. 2, pp. 419-444, 2002.

[23] E. Alpaydin, *Introduction to Machine Learning*.: MIT Press, 2004.

[24] R. Perersen and M. Schoeberl, "An Embedded Support Vector MAchine," in WISES, 2006.

[25] S. Haykin, "Neural Networks and Learning Machines", 3rd ed. United States of America: Pearson Education Inc., 2009.

[26] A.K. Seewald and F. Kleedorfer, "Lambda Pruning: an approximation of the string subsequence kernel for practical SVM classification and redundancy clustering," in *ADAC,* vol. 1, no. 3, pp. 221-239, 2007.

[27] J. Doucette, A.R. McIntyre, P. Lichodzijewski, and M.I. Heywood, "Symbiotic Coevolutionary Genetic Programming: A Benchmarking Study Under Large Attribute Spaces," in journal of *Genetic Programming and Evolvable Machines*, vol. 13, no. 1, pp. 71-101, 2012.

[28] M. Brameier and W. Banzhaf, "A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining," *IEEE Transaction on Evolutionary Computation*, vol. 5, no. 1, pp. 17-26, 2001.

[29] J.R. Quinlan, *C4.5: Programs for Machine learning*.: Morgan Kaufmann, 1993.

[30] Conficker Domain List. [Online]. http://www.malwaredomains.com/wordpress/?cat=111

[31] Conficker Domain List. [Online]. http://net.cs.uni-bonn.de/uploads/media/c_domains_april2009.zip

[32] Kraken Domain List [Online]. http://dvlabs.tippingpoint.com/blog/2008/04/28/owning-kraken-zombies

[33] Kraken Domain List [Online]. http://www.malwaredomains.com/wordpress/?tag=kraken

[34] Paul Royal, "On Kraken and Bobax Botnets," Damballa, 2008. [Online]. https://www.damballa.com/downloads/press/Kraken_Response.pdf

[35] F. Leder and T. Warner, "Know Your Enemy: Containing Conficker To Tame a Malware," Honeynet Project, 2009.

[36] A.K.K. Reddy, "Detecting Networks Employing Algorithmically Generated domain Names," Texas A&M University, Master of Science Thesis 2010.

[37] Alexa. [Online]. http://www.alexa.com/topsites

[38] P. Royal, "Maliciousness in Top-ranked Alexa Domains", [Online]. https://www.barracudanetworks.com/blogs/labsblog?bid=2438

[39] P. Lichodzijewski and M.I. heywood, "Coevolutionary Bid-Based Genetic Programming for Problem Decomposition in Classification," in *Genetic Programming and Evolvable Machines*, vol. 9, no. 4, pp. 331-365, 2008.