Evolving Smart URL Filter in a Zone-Based Policy Firewall for Detecting Algorithmically Generated Malicious Domains

Konstantinos Demertzis(✉) and Lazaros Iliadis

Democritus University of Thrace, 193 Pandazidou st., 68200 Orestiada, Greece
{kdemertz,liliadis}@fmenr.duth.gr

Abstract. Domain Generation Algorithm (DGA) has evolved as one of the most dangerous and "undetectable" digital security deception methods. The complexity of this approach (combined with the intricate function of the fast-flux "botnet" networks) is the cause of an extremely risky threat which is hard to trace. In most of the cases it should be faced as zero-day vulnerability. This kind of combined attacks is responsible for malware distribution and for the infection of Information Systems. Moreover it is related to illegal actions, like money mule recruitment sites, phishing websites, illicit online pharmacies, extreme or illegal adult content sites, malicious browser exploit sites and web traps for distributing virus. Traditional digital security mechanisms face such vulnerabilities in a conventional manner, they create often false alarms and they fail to forecast them. This paper proposes an innovative fast and accurate evolving Smart URL Filter (eSURLF) in a Zone-based Policy Firewall (ZFW) which uses evolving Spiking Neural Networks (eSNN) for detecting algorithmically generated malicious domains names.

Keywords: Domain generation algorithm · Fast-Flux · Evolving spiking neural network · Botnet · Feed forward neural network · Particle swarm optimization

1 Introduction

The most common malware types, aim for the recovery of communication with the Command & Control (C2RS) remote servers and its retention on a regular basis. This is done in order for the botmasters to gather or distribute information and upgrades towards the undermined devices (bots). This communication is usually achieved with the use of hardcoded address or with pool addresses which are controlled by the malware developer or by the botmaster. Modern programming techniques offer malware developers, the chance to use thousands alternating IP addresses of different subnet in order to communicate with the C2RS. However it is easy for the network engineers to trace these IPs, to put them in blacklists.

It is a fact that most recent malware generations use new extremely complicated and smarter ways to communicate with the C2RS under the framework of botnets. More specifically, the communication is achieved by the use of custom distributed

A. Gammerman et al. (Eds.): SLDS 2015, LNAI 9047, pp. 223–233, 2015.
DOI: 10.1007/978-3-319-17091-6_17

dynamic DNS services which are executed in high port numbers. This is done in order to avoid to be traced by security programs located in the gateway of networks. In this way fast-flux botnets are created, whose target is the mapping of a fully qualified domain name to hundreds of IP addresses. These IPs are interchanged too fast, with a combination of random IP addresses and a very small Time-To-Live (TTL) for each partial DNS Resource Record. In this way a domain name can change its corresponding IP address very often (e.g. every 3 minutes). Another usual approach is the blind proxy redirection (BPR) technique. The BPR continuously redirects the applications received by frontend systems to backend servers, in order to spoil the traces and the data that designate an attack. In this way the complexity of the botnets increases [1].

DGA or "*Domain Fluxing*" is the most recent and smarter technique used by the creators of sophisticated malware. This method increases the complexity of the networks exponentially. Its algorithm creates a big number of domain names with specific specification characteristics that can be traced only by their creator. Some of these domains are the actual contact points with the C2RS of the botnets and they are used on a temporal basis and for minimum time intervals. This makes them much more difficult to be spotted. For example the following algorithm 1 creates domains according to a DGA perspective, where each domain is determined based on a date.

Algorithm 1. Generates a domain by the current date [2]

```
1. defgenerate_domain(year, month, day):
2.     """Generates a domain by the current date"""
3.     domain = ""
4. for i in range(32):
5.     year = ((year ^ 8 * year) >> 11) ^ ((year &amp;amp; 0xFFFFFFF0) << 17)
6.     month = ((month ^ 4 * month) >> 25) ^ 9 * (month &amp;amp; 0xFFFFFFF8)
7.     day = ((day ^ (day << 13)) >> 19) ^ ((day &amp;amp; 0xFFFFFFFE) << 12)
8.     domain += chr(((year ^ month ^ day) % 25) + 97)
9.     domain += '.com'
10. return domain
```

E.g., on June 18th, 2014, this method would generate the following domain names:

k.com	*kafph.com*	*kafphogvi.com*	*kafphogvifahu.com*
ka.com	*kafpho.com*	*kafphogvif.com*	*kafphogvifahut.com*
kaf.com	*kafphog.com*	*kafphogvifa.com*	*kafphogvifahutb.com*
kafp.com	*kafphogv.com*	***kafphogvifah.com***	*kafphogvifahutbl.com*

Every time that the botmaster wishes to contact the bots in a predefined strike-time, it creates the proper DNS records for one of the newly created names in the C2RS. In the above example, the strike time is the 18th of June 2014. At this point DNS records are activated only for the domain name ***kafphogvifah.com***. Some minutes before the strike-time the C2RS are activated in order to establish communication. Right afterwards, the botmaster deletes the registrations of the DNS service and deactivates the

C2RS. The thousands domains created by the DGA algorithm daily, the huge complexity of the fast-flux botnets and the distributed function of the C2RS for some minutes per day, make designation of these servers a very tedious task [3].

This research paper proposes the development of an innovative system, capable of protecting from fast-flux botnets that use domain names created with the DGA methodology. Existing methods focus in DNS traffic analysis [4] [5] [6]. This paper proposes an Evolving Smart URL Filter in a Zone-based Policy Firewall for detecting Algorithmically Generated Malicious Domains Names. It is a biologically inspired Artificial Intelligence (AI) security technique as it uses evolving Spiking Neural Networks. The eSNN are the third generation neural networks, emulating the function of the human brain in the most efficient way. This research effort is an enhancement of previous [7] [8] [9] [10]. The efficiency of the proposed eSURLF approach has been compared to other evolving and bio-inspired learning methods, namely: Feed Forward Neural Networks using Particle Swarm Optimization (FNNPSO) technique and Gene Expression Programming (GEP).

1.1 Literature Review

Gu et al. (2007) [11] proposed a method to detect the infection and coordination dialog of botnets by matching a state-based infection sequence model. Based on URL extraction from spam mail Ma, et al. [12], studied a number of machine learning methods for classification web site. Characteristics, such as IP addresses, whois records and lexical features of phishing URLs have been analyzed by McGrath and Gupta [13]. Xie et al [14] in focus on detecting spamming botnets by developing regular expression based signatures from a dataset of spam URLs. Etienne, et al. [15] proposed a technique for the detection and mitigating botnet infection on a network. They use multiple features from DNS queries such as A and NS Records, IP ranges, TTL and alphanumeric characters from domains. In their experiment, they applied Naive Bayesian. Nhauo et al. [16] proposed a method for Classification of Malicious Domains using Support Vector Machine and Bi-gram algorithm using only domain names and their results showed that features extracted by bi-gram were performing a lot better than single alphanumeric character. Antonakakis et. al. [17] uses a combination of clustering and classification algorithms to detecting the DGA-Based Malware. Zhao et al. [18] select a set of attributes from the network flows and then applies a Bayes network and a decision tree algorithm to classify malicious traffic. Also, a number of approaches have been studied in [19] [20] [21] [22] which use the spam emails as the primary information source, for detecting fast-flux domains.

2 URL Filtering

URL filtering is a technique which controls network traffic, by allowing or forbidding access to specific websites based on the information contained in the requested URL. Filters can be applied in various ways such as software, proxy servers, DNS services and firewalls. The ZFW method is an advanced technique used in modern firewalls. It

functions in a flexible way employing control zones. Each zone controls traffic based on the IP and the redirection of the requests from one zone to the other, according to the rules that have been defined for each security level by the system administrator [23]. ZFW offers URL filtering in the following manner:

- ***Local URL List*** which can include URLs or IP addresses that have been traced as risky (black list records) and URLs in which the user wishes to have access (white list records). Before the use of a URL it is checked if it is included in one of the black or white lists and access is forwarded or stopped. The difficulty in regular updating this lists and the non-spotting of zero-day threats are the main disadvantages.
- ***URL Filter Servers*** containing information related to the malware content, which are updated regularly. Before using a URL an HTTP request is sent to the URL filter server in order to be checked and to allow or restrict access. If the URL filter server is not responding the next one is searched. A serious drawback is the potential that filter servers do not respond whereas there is no mechanism for tracing zero-day threats.
- ***Local URL lists with URL filter servers***. It is a hybrid model of the above two cases that combines advantages and disadvantages.

3 Datasets

Two datasets namely the *dga_timestamp* and the *dga_dictionary* were constructed and used for testing. As legit domains 100,000 domain names were used. They were chosen randomly from the database with the 1 million most popular domain names of Alexa [24]. For the malicious domains the updated list of the Black Hole DNS database was used [25]. This list includes 16,374 records from domains that have been traced and characterized as dangerous. More over in the *dga_time stamp* dataset 15,000 domain name records were added labeled as malicious. They were created based on the time stamp DGA algorithm, with length from 4 to 56 characters of the form ***18cbth51n205gdgsar1io1t5.com***. In the *dga_dictionary* dataset 15,000 domain name records were added labeled as malicious, which were created with the use of words of phrases coming from an English dictionary. Their length varied from 4 to 56 characters of the form ***hotsex4rock69burningchoir.com***. In both of the cases the characteristics used as independent parameters were: ***length*** (the length of the strings of the domains) ***entropy*** (the entropy of each domain as degree of uncertainty, with the higher values met in the DGA domains), ***alexa_grams*** (the degree of coherence between the domain and the list of domains originating from Alexa. This is done with the technique of the probability linguistic model for the forecasting of the next n-gram element), ***word_grams*** (the degree of coherence between the domain and a list of 479,623 words or widely used characters. It is estimated with the same method as in the previous one), ***differences*** (the difference between the values of alexa_grams and word_grams). The depended variable was the ***class*** of the domain (*legit* or *malicious*) [26]. Duplicate records and records with incompatible characters were removed. Also the outliers and the extreme values spotted were removed based on the Inter Quartile

Range (IQR) technique [27]. After this preprocessing operation the *dga _time stamp* dataset was left with 116,756 records from which the 90,338 are class *legit* and the rest 26,418 are class *dga*, whereas the *dga_dictionary* dataset includes 116,071 records from which the 90,445 are class *legit* and the rest 25,626 class *malicious*.

4 Methods and Materials

4.1 Evolving Spiking Neural Network

The eSNNs based on the "Thorpe" neural model [28] are modular connectionist-based systems that evolve their structure and functionality in a continuous, self-organized, on-line, adaptive, interactive way from incoming information [29]. In order to classify real-valued data sets, each data sample, is mapped into a sequence of spikes using the Rank Order Population Encoding (ROPE) technique [30] [31]. In this encoding method neurons are organized into neuronal maps which share the same synaptic weights. Whenever the synaptic weight is modified, the same modification is applied to the entire population of neurons within the map. Inhibition is also present between each neuronal map. If a neuron spikes, it inhibits all the neurons in the other maps with neighboring positions. This prevents all the neurons from learning the same pattern. When propagating new information, neuronal activity is initially reset to zero. Then, as the propagation goes on, each time one of their inputs fire, neurons are progressively desensitized. This is making neuronal responses dependent upon the relative order of firing of the neuron's afferents [32] [33]. Also in this model the neural plasticity is used to monitor the learning algorithm by using one-pass learning method. The aim of this learning scheme is to create a repository of trained output neurons during the presentation of training samples [34].

4.2 Feed Forward Neural Networks and Heuristic Optimization Methods

A common form of ANN are the feed forward ones (FNN) with three layers (Input, Hidden and Output). A typical training process includes the following steps [35]:

- The weighted sums of the inputs are calculated based on function (1):

$$s_j = \sum_{i=1}^{n} \left(W_{ij} X_i \right) - \theta_j \quad j=1,2,...,h \tag{1}$$

where *n,h,m* are the number of input, hidden and output nodes respectively and *Wij* is the connection weight from the *i*th node of the input layer to the *j*th node of the hidden layer. Also θ_j *is the* bias (threshold).

- The output for each hidden node is estimated with the following equation (2):

$$S_j = sigmoid_{(s_j)} = \frac{1}{\left(1+exp(-s_j)\right)} j=1,2,...,h \tag{2}$$

- The final output is estimated based on the equations (3) and (4) below:

$$o_k = \sum_{j=1}^{h}\left(W_{jk}S_j\right) - \theta'_k \quad k=1,2,...,m \tag{3}$$

$$O_k = sigmoid_{(o_k)} = \frac{1}{\left(1+exp(-o_k)\right)} k=1,2,...,m \tag{4}$$

Various heuristic optimization methods have been used to train FNNs such as Genetic Algorithms (GAs) and Particle Swarm Optimization (PSO) algorithms. Generally, there are three heuristic approaches used to train or optimize FNNs. First, heuristic algorithms are used to find a combination of weights and biases which provide the minimum error for a FNN. Second, heuristic algorithms are employed to find the optimal architecture for the network related to a specific case. The last approach is to use an evolutionary algorithm to tune the parameters of a gradient-based learning algorithm, such as the learning rate and momentum [36]. In this research effort PSO has been employed to provide the optimal FNN model.

4.2.1 Particle Swarm Optimization

PSO is an evolutionary computation technique which is proposed by Kennedy and Eberhart. It was inspired by social behavior of bird flocking or fish schooling. PSO shares many similarities with evolutionary computation techniques such as GA. The system is initialized with a population of random solutions and searches for optima by updating generations. However, unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. Each particle keeps track of its coordinates in the problem space which are associated with the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called "*pbest*". Another "best" value that is tracked by the particle swarm optimizer is the best value obtained so far by any particle in its neighbors. This location is called "*lbest*". When a particle takes all the population as its topological neighbors, the best value is a global best and is called "*gbest*". According to the PSO at each time step there is a change in the velocity (acceleration) of each particle toward its *pbest* and *lbest* locations (local version of PSO). Acceleration is weighted by a random term, with separate random numbers being generated for acceleration toward *pbest* and *lbest* locations. PSO was mathematically modeled as follows [36]:

$$v_i^{t+1} = wv_i^t + c_1 \times rand \times \left(pbest_i - x_i^t\right) + c_2 \times rand \times \left(gbest\text{-}x_i^t\right) \tag{5}$$

$$x_i^{t+i} = x_i^t + v_i^{t+i} \tag{6}$$

Where v_i^t is the velocity of particle i at iteration t, w is a weighting function, c_j is an acceleration coefficient, *rand* is a random number between 0 and 1, x_i^t is the current position of particle i at iteration t, $pbest_i$ is the pbest of agent i at iteration t, and *gbest* is the best solution so far. Where wv_i^t, provides exploration ability for PSO, the

$$c_1 \times rand \times \left(pbest_i - x_i^t\right) + c_2 \times rand \times \left(gbest\text{-}x_i^t\right) \tag{7}$$

represent private thinking and collaboration of particles. The PSO starts by randomly placing the particles in a problem space. During each iteration, the velocities of particles are calculated using equation (5). After defining the velocities, the positions of particles can be calculated as shown in equation (6). The process of changing particles' positions continues until an end criterion is met.

4.3 Gene Expression Programming

The GEP is an evolutionary algorithm that uses populations of individuals and chooses the optimal ones based on a fitness function. Then it imports new individuals or potential solutions in the population, by using one or more genetic operators. The substantial difference between the GEP and the GA and Genetic Programming (GP) is in the nature of the individuals. Specifically, GA uses a sequence of symbols of stable length whereas GP uses nonlinear entities (parse trees) of various length and shape. GEP employs both above forms of individuals. Initially they are encoded like linear strings of stable length (genotype of chromosome) and then they are expressed as expression trees (EXTR) with different shape and size (phenotype). The correlation between the chromosomes and the EXTR in GEP declares an exclusive translation system from the language of the chromosomes to the one of the EXTR. The set of the genetic operators applied to GEP and import new individuals they always create syntactically correct EXTR [37].

5 Description of the Proposed Method

The proposed herein methodology uses an eSNN classification approach in order to detect and verify the DGA domain names. The topology of the developed eSNN is strictly feed-forward, organized in several layers and weight modification occurs on the connections between the neurons of the existing layers. The encoding is performed by ROPE technique with 20 Gaussian Receptive Fields (GRF) per variable. The data are normalized to the interval [-1, 1] and so the coverage of the Gaussians is determined by using *i_min* and *i_max*. Each input variable is encoded independently by a group of one-dimensional GRF. The GRF of neuron i is given by its center μ_i by equation (8) and width σ by equation (9)

$$\mu_i = I^n_{min} + \frac{2i-3}{2}\frac{I^n_{max} - I^n_{min}}{M-2} \tag{8}$$

$$\sigma = \frac{1}{\beta}\frac{I^n_{max} - I^n_{min}}{M-2} \tag{9}$$

where $1 \leq \beta \leq 2$ and the parameter β directly controls the width of each Gaussian receptive field. When a neuron reaches its threshold, it spikes and inhibits neurons at equivalent positions in the other maps so that only one neuron will respond at any location. Every spike triggers a time based Hebbian-like learning rule that adjusts the synaptic weights. For each training sample i with class label l which represent a *legit*

domains, a new output neuron is created and fully connected to the previous layer of neurons, resulting in a real-valued weight vector $w^{(i)}$ with $w_j^{(i)} \in R$ denoting the connection between the pre-synaptic neuron j and the created neuron i. In the next step, the input spikes are propagated through the network and the value of weight $w_j^{(i)}$ is computed according to the order of spike transmission through a synapse

$$j: w_j^{(i)} = (m_l)^{order(j)} \tag{10}$$

where j is the *p*re-synaptic neuron of i. Function *order(j)* represents the rank of the spike emitted by neuron j. The firing threshold $\theta^{(i)}$ of the created neuron i is defined as the fraction $c_l \in R$, $0 < c_l < 1$, of the maximal possible potential

$$u_{max}^{(i)}: \theta^{(i)} \leftarrow c_l u_{max}^{(i)} \tag{11}$$

$$u_{max}^{(i)} \leftarrow \sum_j w_j^{(i)} (m_l)^{order(j)} \tag{12}$$

The weight vector of the trained neuron is compared to the weights corresponding to neurons already stored in the repository. Two neurons are considered too "similar" if the minimal *Euclidean* distance between their weight vectors is smaller than a specified similarity threshold s_l. Both the firing thresholds and the weight vectors were merged according to equations (13) and (14):

$$w_j^{(k)} \leftarrow \frac{w_j^{(i)} + N w_j^{(k)}}{1+N} \tag{13}$$

$$\theta^{(k)} \leftarrow \frac{\theta^{(i)} + N\theta^{(k)}}{1+N} \tag{14}$$

Integer N denotes the number of samples previously used to update neuron k. The merging is implemented as the average of the connection weights, and of the two firing thresholds. After merging, the trained neuron i is discarded and the next sample processed. If no other neuron in the repository is similar to the trained neuron i, the neuron i is added to the repository as a new output.

6 Results

The performance of the employed algorithms for the case of the *dga_dictionary* dataset has been quite high as the obtained correlation shows. On the other hand, in the *dga_timestamp* dataset there was quite a lot of noise, due to the unstructured text of domain names that cannot be easily understood by machines. Regarding the overall efficiency of the methods, the results show that the eSNN has much better generalization performance and more accurate classification output. The accuracy comparison of the evolving algorithms with 10-fold cross validation is shown in table 1 and the confusion matrices in tables 2, 3 and 4

Table 1. Accuracy (ACC) Comparison between FFNN PSO, GEP and eSNN

	dga_dictionary dataset	**dga _timestamp dataset**
Classifier	**ACC**	**ACC**
FNNPSO	95.5%	91.9%
GEP	92.1%	90.6%
eSNN	**95.8%**	**92.4%**

Table 2. Confusion matrix for FNNPSO

CONFUSION MATRIX FOR FFNNPSO						
	dga_dictionary dataset			**dga _timestamp dataset**		
	Legit	DGA	Accuracy	Legit	DGA	Accuracy
Legit	**86556**	3889	95,7%	**84105**	6233	93,1%
DGA	1204	**24422**	95,3%	2457	**23961**	90,7%
	Overall Accuracy		**95.5%**	**Overall Accuracy**		**91.9%**

Table 3. Confusion matrix for GEP

CONFUSION MATRIX FOR GEP						
	dga_dictionary dataset			**dga _timestamp dataset**		
	Legit	DGA	Accuracy	Legit	DGA	Accuracy
Legit	**83933**	6512	92,8%	**81666**	8672	90,4%
DGA	2204	**23422**	91,4%	2430	**23988**	90,8%
	Overall Accuracy		**92.1%**	**Overall Accuracy**		**90.6%**

Table 4. Confusion matrix for eSNN

CONFUSION MATRIX FOR eSNN						
	dga_dictionary dataset			**dga _timestamp dataset**		
	Legit	DGA	Accuracy			
Legit	**87732**	2713	97%	**84647**	5691	93,7%
DGA	1384	**24242**	94,6%	2351	**24067**	91,1%
	Overall Accuracy		**95.8%**	**Overall Accuracy**		**92.4%**

7 Discussion – Conclusions

An innovative biologically inspired artificial intelligence computer security technique has been introduced in this paper. An evolving Smart URL Filter in a Zone-based Policy Firewall proposed for detecting Algorithmically Generated Malicious Domains Names. It performs classification by using eSNN for the detection of DGA with high accuracy and generalization. The classification performance and the accuracy of the eSNN model were experimentally explored based on different datasets and compared with other evolving algorithms and reported very promising results. In this way it adds a higher degree of integrity to the rest of the security infrastructure of a ZFW.

As a future direction, aiming to improve the efficiency of the proposed method, it would be essential to try feature minimization using Principal Component Analysis or other existing approaches. Also additional computational intelligence methods could be explored and compared on the same security task. Finally the eSURLF could be improved towards a better online learning with self-modified parameter values.

References

1. www.damballa.com
2. www.crowdstrike.com
3. DGAs and Cyber-Criminals: A Case Study, Research Note. www.damballa.com
4. Yadav, S., Reddy, A.K.K., Reddy, A.L.N., Ranjan, S.: Detecting Algorithmically Generated Domain-Flux Attacks With DNS Traffic Analysis. ACM **20**(5) (2012)
5. Perdisci, R., Corona, I., Giacinto, G.: Early Detection of Malicious Flux Networks via Large-Scale Passive DNS Traffic Analysis. By the IEEE Computer Society (2012)
6. Bilge, L., Kirda, E., Kruegel, C., Balduzzi, M.: EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis. TISSEC **16**(4), Article No. 14 A (2014)
7. Demertzis, K., Iliadis, L.: A hybrid network anomaly and intrusion detection approach based on evolving spiking neural network classification. In: Sideridis, A.B. (ed.) E-Democracy 2013. CCIS, vol. 441, pp. 11–23. Springer, Heidelberg (2014)
8. Demertzis, K., Iliadis, L.: Evolving computational intelligence system for malware detection. In: Iliadis, L., Papazoglou, M., Pohl, K. (eds.) CAiSE Workshops 2014. LNBIP, vol. 178, pp. 322–334. Springer, Heidelberg (2014)
9. Demertzis, K., Iliadis, L.: Bio-Inspired hybrid artificial intelligence framework for cyber security. In: Proceedings of the 2nd Conference on CryptAAF, Athens, Greece (2014)
10. Demertzis, K., Iliadis, L.: Bio-Inspired Hybrid Intelligent Method for Detecting Android Malware. In: Proceedings of the 9th KICSS Conference, Limassol, Cyprus (2014)
11. Gu, G., Porras, P., Yegneswaran, V., Fong, M., Lee W.: Bothunter: detecting malware infection through ids-driven dialog correlation. In: 16th USENIX, pp. 1--16 (2007)
12. Ma, J.: Beyond blacklist: learning to detect malicious website from suspicious URLs. In: SIGKDD Conference, Paris, France (2009)
13. McGrath, D.K., Gupta, M.: Behind phishing: an examination of phisher modi operandi. In: USENIX on Large-scale Exploits and Emergent Threats (LEET) (2008)
14. Xie, Y., Yu, F., Achan, K., Panigrahy, R., Hulten, G., Osipkov, I.: Spamming botnets: signatures and characteristics. ACM SIGCOMM Comp. Comm. Review (2008)
15. Stalmans, E.: A framework for DNS based detection and mitigation of malware infections on a network. In: Information Security South Africa Conference (2011)
16. Nhauo, D., Sung-Ryul, K.: Classification of malicious domain names using support vector machine and bi-gram method. J. of Security and its Applications **7**(1) (2013)
17. Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou, N., Abu, S., Lee, W., Dagon, D.: From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware (2012)
18. Zhao, D., Traore, I., Sayed, B., Lu, W., Saad, S., Ghorbani, A.: Botnet detection based on traffic behavior analysis and flow intervals. J. Computer Security **39,** 2–16 (2013)
19. Holz, T., Gorecki, C., Rieck, K., Freiling, F.: Measuring and detecting fast-flux service networks. In: Network & Distributed System Security Symposium, NDSS 2008 (2008)
20. Passerini, E., Paleari, R., Martignoni, L., Bruschi, D.: Fluxor: detecting and monitoring fast-flux service networks. In: DIMVA 2008 (2008)

21. Nazario, J., Holz, T.: As the net churns fast-flux botnet observations. In: MALWARE (2008)
22. Konte, M., Feamster, N., Jung, J.: Dynamics of online scam hosting infrastructure. In: Passive and Active Measurement Conference, PAM 2009 (2009)
23. Cisco Router and Security Device Manager 2.4 User's Guide. www.cisco.com
24. http://www.alexa.com/
25. http://www.malwaredomains.com/
26. https://www.clicksecurity.com/
27. Upton, G., Cook, I.: Understanding Statistics. Oxford University Press, p. 55 (1996)
28. Thorpe, S.J., Delorme, A., Rullen, R.: Spike-based strategies for rapid processing (2001)
29. Schliebs, S., Kasabov, N.: Evolving spiking neural network—a survey. Springer (2013)
30. Delorme, A., Perrinet, L., Thorpe, S.J.: Networks of Integrate-and-Fire Neurons using Rank Order Coding. Pub. in Neurocomputing **38-40**(1-4), 539–545 (2000)
31. Thorpe, S.J., Gautrais, J.: Rank order coding. In: CNS 1997: 6th Conf. on Computational Neuroscience: Trends in Research, pp. 113–118. Plenum Pr. (1998)
32. Kasabov, N.: Evolving connectionist systems: Methods and Applications in Bioinformatics, Brain study and intelligent machines. Springer (2002)
33. Wysoski, S.G., Benuskova, L., Kasabov, N.: Adaptive learning procedure for a network of spiking neurons and visual pattern recognition. In: Blanc-Talon, J., Philips, W., Popescu, D., Scheunders, P. (eds.) ACIVS 2006. LNCS, vol. 4179, pp. 1133–1142. Springer, Heidelberg (2006)
34. Schliebs, S., Defoin-Platel, M., Kasabov, N.: Integrated feature and parameter optimization for an evolving spiking neural network. In: Köppen, M., Kasabov, N., Coghill, G. (eds.) ICONIP 2008, Part I. LNCS, vol. 5506, pp. 1229–1236. Springer, Heidelberg (2009)
35. Iliadis, L.: Intelligent Information Systems and applications in risk estimation. A. Stamoulis publication, Thessaloniki (2008) ISBN: 978-960-6741-33-3
36. Mirjalili, S., Hashim, S., Sardroudi, H.: Training feedforward neural networks using hybrid particle swarm optimization and gravitational search algorithm. Elsevier (2012)
37. Ferreira, C.: Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence, 2nd edn., Springer (2006)