Learning From Evolving Network Data for Dependable Botnet Detection

Duc C. Le, Nur Zincir-Heywood
Faculty of Computer Science, Dalhousie University,
Halifax, Nova Scotia, Canada
lcd@dal.ca, zincir@cs.dal.ca

***Abstract*—This work presents an emerging problem in real-world applications of machine learning (ML) in cybersecurity, particularly in botnet detection, where the dynamics and the evolution in the deployment environments may render the ML solutions inadequate. We propose an approach to tackle this challenge using Genetic Programming (GP) - an evolutionary computation based approach. Preliminary results show that GP is able to evolve pre-trained classifiers to work under evolved (expanded) feature space conditions. This indicates the potential use of such an approach for botnet detection under non-stationary environments, where much less data and training time are required to obtain a reliable classifier as new network conditions arise.**

***Index Terms*—botnet detection, machine learning, evolving networks**

I. Introduction

Cybercrime has evolved drastically along with the development of the Internet and computer systems since the early days. Hence, cybersecurity is one of the most important aspects in managing networks and connected systems nowadays. In the recent years, many machine learning (ML) based intrusion detection solutions have been proposed for dealing with the enormous amount of data and a wide variety of network threats [1].

A particular challenge in the applications of ML in intrusion detection is to maintain the reliability / robustness of ML based systems under dynamic and evolving environments [2]–[4]. Different environmental changes may appear with newly introduced technologies and/or novel threats. Among these, many variations may emerge in the network / service environments and data collection / processing procedures, that could negatively affect the performances of deployed intrusion detection systems. Examples of such changes include:

- Variations in network infrastructures, e.g. new network devices / types, topology changes that provide new information to intrusion detection systems;
- New or updated monitoring / data capturing systems that allow extracting more features from current data streams;
- Concept shift and drift in normal network behaviours and newly discovered attacks and malicious network traffic.

Essentially, the changes may affect ML based intrusion detection systems via the feature space (input space), output space, and / or objective function. In this paper, we focus on a type of change: feature space expansion of network data, where new information (features) are introduced from time to time and need to be learned from / incorporated to ML models. Under such conditions, the model needs to maintain, or even improve the intrusion detection performance. Employing an evolutionary computation method, namely Genetic Programming (GP), this work aims to evaluate the possibility of evolving pre-trained GP classifiers to work under the aforementioned changes of the deployment environments. In doing so, we aim to improve the robustness and reliability of an ML based network security system, and to reduce training time and increase training data efficiency under new network conditions.

The rest of the paper is organized as follows: Section II summarizes recent developments of ML techniques reported in the literature for network intrusion detection. Section III formulates the problem in the context of botnet detection using network traffic flows. The section also describes the employed GP method and the introduced technique, which allows the pre-trained GP classifiers to evolve under new data conditions. Section IV presents the experiments and evaluation results. Finally, conclusions are drawn and the future work is discussed in Section V.

II. Related Work

In recent years, ML techniques have been applied extensively in cybersecurity in general and intrusion detection in particular [1]–[3]. Different ML techniques for botnet detection have been explored in several works in the literature, including decision trees [4], [5], neural networks [6], [7], and unsupervised learning approaches [8]–[10]. Genetic programming, an evolutionary computation approach, has been successfully applied in several works as well [11]–[16]. Song et al. in [11] applied Linear GP for intrusion detection on a large and highly imbalanced dataset (KDD-99). Haddadi et al. employed Symbiotic Bid-based GP for botnet detection and showed the advancement of GP over rule based and packet payload inspection based systems [12]. In [15], Sen and Clark applied GP and grammatical evolution for intrusion detection in mobile network environments, with a focus on power efficiency of the solution. In general, Wu and Banzhaf surveyed applications of computational intelligence, from artificial neural networks, fuzzy systems, evolutionary computation, artificial immune systems, to swarm intelligence, and soft computing in intrusion detection [16].

978-3-903176-24-9 © 2019 IFIP

Some particular changes in deployment environments of ML based intrusion detection systems have been addressed in the literature. For addressing concept shift and drift, online stream learning techniques have been employed in [13], [14], [17]. Similarly, intrusion detection systems on "novel" data classes have been addressed partially in [4], [17]. In [4], Haddadi et al. examined the effectiveness of ML methods for botnet detection under botnet evolution. However, changes affecting the feature space of ML solutions for cybersecurity have not been addressed in detail. In a related work, [18], Manzoor et al. proposed an outlier detection technique for feature-evolving data streams using streaming random projection and ensemble of half-space chains. Different from the work in the literature, in this work, our main focus is addressing the challenge of feature space expansion (evolution) using an evolutionary computation technique, GP, for botnet detection systems.

III. Methodology

As discussed in Section I, this work focuses on learning under feature space expansion of network data for robust and reliable botnet detection. To achieve this, different methods are introduced based on a Linear GP approach in order to enable and/or accelerate incorporating newly introduced (evolved) features to pre-trained GP solutions. In this work, the data is represented as network traffic flows. A network traffic flow is a summarization of a "connection", i.e. a sequence of aggregated packets between a pair of network devices. A network traffic flow is commonly identified by a set of five different attributes (5-tuples) including source and destination IP addresses and port numbers, and the protocol, over a predetermined duration. Network traffic flow is a common data type for ML applications in cybersecurity in general and botnet detection in particular [1], [19].

A. Problem Formulation

Different changes in network environment and data collection / data processing steps may result in different feature spaces of ML application in cybersecurity. Particularly, in network traffic flow data, examples of such changes are updated configurations for exporting flows, or new flow exporter, which may introduce new features (information) in the exported flows. To formulate the problem, we assume that in deployment is a ML based botnet detection system, $\mathcal{D}$, working on the original feature space, $\mathcal{F}$, i.e on data points $x \in \mathbb{R}^N$, $|\mathcal{F}| = N$. At some point, the aforementioned changes may result in new information / features, hence maybe a larger feature space: $\mathcal{F}'$, $\mathcal{F} \subset \mathcal{F}'$. To traditional ML methods, which assumes a fixed feature space, the newly introduced features provide no benefit. Hence, new models need to be trained with additional time, effort, and training examples to ensure an adequate performance on the new data which have new features. This work focuses on addressing the challenge of evolving the detection system (classifier), $\mathcal{D}$, into an updated classifier $\mathcal{D}'$ that works on the expanded feature space $\mathcal{F}'$. This is to exploit the knowledge – that is embedded in $\mathcal{D}$ through its training process – to quickly obtain a working classifier for $\mathcal{F}'$ at the production level. To accommodate the changes without generating and/or training new ML models, we need to address the challenge of recognizing and learning from newly emerging features, $\mathcal{F}' \setminus \mathcal{F}$. Moreover, while doing this, we need to maintain the current performance on the known features and classes.

B. Linear Genetic Programming (LGP)

LGP is a GP variant where programs (solutions) in a population (set of individual programs) are represented as a linear sequence of instructions [20]. The execution of a LGP program follows a graph-based data flow. Each instruction is executed based on a defined arithmetic operation and operands. The operands can be registers, constants, or input values. The output is taken at the end of the program as the values of designated registers. By manipulating the content of the registers, which may carry the input feature values, LGP potentially has the ability to perform feature construction [20].

LGP is trained using data subsets through a number of generations. In each generation, the programs in the GP population are evaluated on the training subset. Then, well performed programs in the population are selected for variation operators to replace the worst ranked individuals and evolve the population in the next generation. The variation operators include: (i) Crossover, where blocks of instructions are swapped between pairs of parent individuals; (ii) Micro mutation, where a part of a selected instruction is modified (target register, operands, or operator); and (iii) macro mutation, where a selected instruction is replaced, deleted, or a random instruction is inserted.

C. Learning From New Features

LGP has two unique advantages allowing a solution for learning from expanding / evolving feature space: (i) By its nature, LGP is a population based approach, where changes in data can be learned gradually through generations; and (ii) Based on the use of input registers in LGP programs, feature space expansion can be accommodated by appropriately extending the input register vector.

In addition to extending the input register vector, variation operators can also be modified to accommodate those new features (input registers). In doing so, our principal objective is to allow LGP programs to evolve on the new feature set using the established structures, i.e. knowledge that has been learned on the old feature set $\mathcal{F}$, without violating them.

The following scheme for modifying the variation operators is explored in this work: Upon seeing an expanded input vector, the variation operators, micro- and macro-mutations in this case, are adjusted to select new features ($\mathcal{F}' \setminus \mathcal{F}$) with higher probability than the existing features. The probability is controlled using two parameters: The initial multiply factor β where ($\beta \geq 1$), and the number of generations, G. As such, the multiply factor for the probability of selecting a new feature in generation g, where $g \geq 0$, is:

$$\beta_g = \begin{cases} (1-\beta) \times g/G + \beta, \text{ if } g \leq G \\ 1, \text{ if } g > G \end{cases} \quad (1)$$

Essentially, this scheme allows a seamless transition between the bias state of prioritizing the selection of new features (in the first G generations) to learning from all features equally (after generation G). This is done by gradually reducing the multiply factor from the maximum value β at generation 0, i.e when new features arrive, to 1 at generation G, i.e. treating all features equally.

IV. Evaluation and Results

A. Dataset

The experiments are performed on a public dataset for botnet detection, the CTU13 dataset [21]. Specifically, we use the Neris botnet network traffic flow capture from the CTU13 dataset. This capture has been demonstrated as one of the most challenging among the captures provided in CTU13 [8], [21]. The evolution in data feature space of the Neris botnet traffic capture is simulated through the use of six basic netflow features (as provided in the dataset) and an extended set of 24 netflow features (maximum number of features achievable, exported from CTU13 provided Argus[1] binary file[2]). This can be seen as an update in real-world monitoring process, where more information is extracted from the same source of data. The original feature set is $\mathcal{F}$ = {*duration, protocol, flags, type of service, #packets, #bytes*}. On the other hand, an updated feature space, $\mathcal{F}'$, exported using expanded Argus configuration includes the following features: $\mathcal{F}' = \mathcal{F} \cup$ {*#src packet, #dst packets, #app bytes, #src bytes, #src app bytes, #dst bytes, #dst app bytes, #load, #src load, #dst load, loss, src loss, dst loss, rate, src mean size, dst mean size, src max size, src min size*} where src: source, dst: destination, and app: application. In each run, the dataset is randomly split four-way into training and test sets under basic feature set $\mathcal{F}$, and under expanded feature set $\mathcal{F}'$. By exemplar count, there are 29,967 normal flows, 2,973 botnet command & control (malicious) flows, and 182,014 botnet (malicious) flows in the dataset.

B. Experiment Settings

This work assumes a binary classification, where the two classes are *normal* (negative) and *botnet* (positive). The botnet detector performance is measured using the following commonly used metrics [1]: Accuracy, Normal and Botnet Detection Rates (NDR and BDR), and Class-wise DR (CDR). Denoting True Positive, True Negative, False Positive, and False Negative as TP, TN, FP, and FN, respectively, we have:

$$Accuracy = (TN + TP) \,/\, (TN + TP + FN + FP), \quad (2)$$

$$NDR = TN \,/\, (TN + FP), \quad (3)$$

$$BDR = TP \,/\, (TP + FN), \quad (4)$$

$$CDR = (NDR + BDR) \,/\, 2. \quad (5)$$

Empirically chosen parameters of LGP are presented in Table I. Multi-objective selection is employed in this work to address two objectives simultaneously: (i) Maximize detection rate (over all classes); and (ii) Maximize accuracy. This not only allows evolving well performed programs in the defined objectives, but also indirectly address the false alarm rate and class detection rate. Multi-objective selection is done through the use of Pareto ranking. In the experiments, after a LGP population $\mathcal{P}^{\mathcal{F}}$ is trained on the original feature space $\mathcal{F}$ for 300 generations, it is used as the initial population for $\mathcal{P}^{\mathcal{F}'}_{\beta}$ to evolve under the expanded feature set $\mathcal{F}'$. In order to examine the effect of β, i.e. different levels of biasing toward the new features in $\mathcal{F}' \setminus \mathcal{F}$ introduced in III-C, we choose the following values $\{1, 2, 4\}$ for β, and 50 for G. These parameters result in the LGP populations $\mathcal{P}^{\mathcal{F}'}_{\beta=1}$, $\mathcal{P}^{\mathcal{F}'}_{\beta=2}$, $\mathcal{P}^{\mathcal{F}'}_{\beta=4}$, respectively. Two other LGP populations are also trained for comparison. The population $\mathcal{P}^{\mathcal{F}'}$ assumes the solutions learned in $\mathcal{P}^{\mathcal{F}}$ and continues to evolve using only features in $\mathcal{F}$, while the population $\mathcal{Q}^{\mathcal{F}'}$ is trained from scratch on $\mathcal{F}'$. Experiments are performed and results are obtained based on 20 runs.

TABLE I
Parameters of LGP

Parameter	Value
Population size	500
Data subset size	200
Crossover rate	0.5
Micro mutation rate	0.8
Macro mutation rate	0.8
Function set	$\{+, -, \times, /, >, \sin, \exp, \log\}$
Maximum program length	100

C. Results

Figure 1 shows the *training* results (Accuracy and CDR) on training data subsets of the populations over the training generations, and Table II shows *test* results of the LGP populations after different number of generations.

Note that the population $\mathcal{P}^{\mathcal{F}}$ achieves nearly identical performance (on test data with feature space $\mathcal{F}$) to $\mathcal{P}^{\mathcal{F}'}_{\beta}$ at 0 generation (Table II). As shown in Figure 1, on small feature space, $\mathcal{F}$, LGP ($\mathcal{P}^{\mathcal{F}}$) struggles to learn to separate normal and botnet flows from the limited number of features. Its accuracy and CDR improve very slowly over the training generations.

Figure 1 clearly shows that when presented with a larger feature space, $\mathcal{F}'$, $\mathcal{P}^{\mathcal{F}'}_{\beta}$ ($\beta = 1,\ 2,\ 4$) are able to maintain the performance that $\mathcal{P}^{\mathcal{F}}$ achieves on $\mathcal{F}$, and evolve from that to take advantage of new features in $\mathcal{F}'$. Results in Table II and Figure 1 show that $\beta = 1$ gives better results, in terms of Accuracy and CDR, than $\beta = 2$ and 4 after almost all training generations. In fact, it appears that a higher β value generates worse results than lower ones (Figure 1). This demonstrates that by simply letting the variation operators select features from the whole feature set $\mathcal{F}'$, the LGP is able to effectively evolve a pre-trained population to a better population for new conditions (in this case, under a larger feature space). Artificially introducing bias towards newly introduced features ($\beta = 2,\ 4$) may actually hinder the natural evolution of a LGP program under the new conditions (use case) presented in this paper. This suggests that a simple bias scheme does

[1]https://qosient.com/argus/

[2]mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-50/

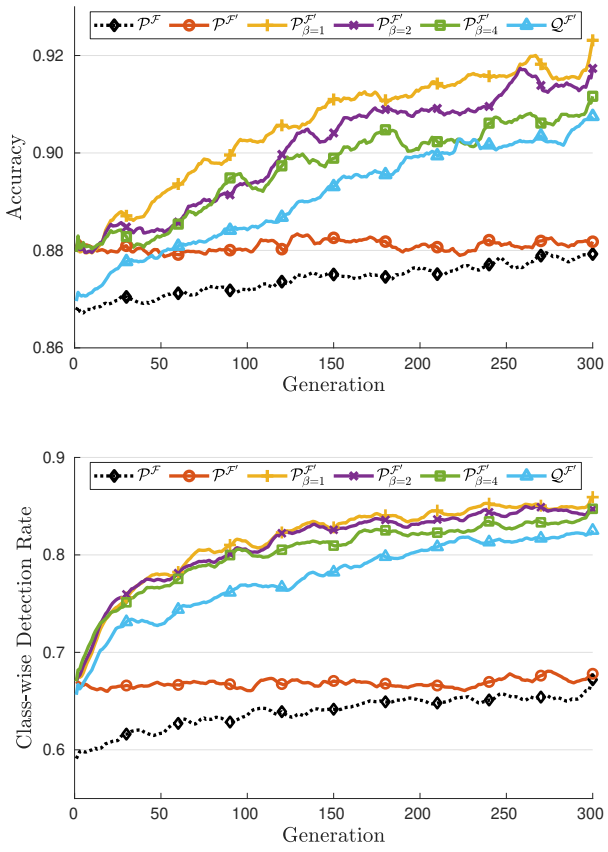

Fig. 1. *Training* accuracy and class-wise detection rates of the populations on $\mathcal{F}'$ training subsets by number of generations

not help and more carefully designed methods are needed to better adapt to feature space expansion.

On the other hand, continuing to evolve the LGP population using the old feature set, as in population $\mathcal{P}^{\mathcal{F}'}$, is proved unbeneficial. Using this strategy, LGP was unable to improve the solution inherited from $\mathcal{P}^{\mathcal{F}}$ in the new environment. A new LGP population training from scratch ($\mathcal{Q}^{\mathcal{F}'}$) on the feature set $\mathcal{F}'$ could quickly surpass $\mathcal{P}^{\mathcal{F}'}$ even after just 20 generations (Table II). This clearly shows that by continuing to evolve on features in $\mathcal{F}$, $\mathcal{P}^{\mathcal{F}'}$ misses critical information introduced in $\mathcal{F}' \setminus \mathcal{F}$.

Finally, comparing $\mathcal{P}_{\beta}^{\mathcal{F}'}$ and $\mathcal{Q}^{\mathcal{F}'}$, it is obvious that $\mathcal{P}_{\beta}^{\mathcal{F}'}$, especially with $\beta = 1$, show better performances than $\mathcal{Q}^{\mathcal{F}'}$ on $\mathcal{F}'$ throughout the training generations. After 50 (100) training generations on $\mathcal{F}'$, $\mathcal{P}_{\beta}^{\mathcal{F}'}$ is able to obtain similar results to $\mathcal{Q}^{\mathcal{F}'}$ at 100 (200) generations. This demonstrates that the previously trained LGP populations can be successfully evolved to work under expanded feature spaces to reduce time, effort, and training examples.

V. Conclusion

Our main objectives in this work were to investigate the capability of a LGP based botnet detector for evolving under expanded feature space conditions. In doing so, the aim is to exploit the learned knowledge embedded in a pre-trained LGP population under the existing feature space, in order to assist the LGP population in quickly learning to work under the new feature space. To this end, a modification scheme to variation operators of LGP is presented. The empirical experiments performed and the results obtained show the potential of such an approach, where it enables a LGP population to continue to evolve under a larger feature set without violating previously learned solutions. Evolving from a pre-trained LGP botnet detector, it is shown that the scheme can reduce the training time and the training data required by half. Future work will further investigate the ability of evolving existing ML solutions to adapt to different changes in cybersecurity environments including but not limited to shrinking feature spaces. Furthermore, the effects of different methods and their parameters for learning under feature space expansion on GP population and behaviours of GP programs will be investigated.

TABLE II
Test results of GP populations trained on $\mathcal{F}'$ after different number of generations.

# gen.	Population	Accuracy	NDR	BDR	CDR
0	$\mathcal{P}_{\beta}^{\mathcal{F}'}$	73.90	53.12	77.27	65.20
20	$\mathcal{P}^{\mathcal{F}'}$	75.77	50.33	79.89	65.11
	$\mathcal{P}_{\beta=1}^{\mathcal{F}'}$	69.53	75.57	68.56	72.06
	$\mathcal{P}_{\beta=2}^{\mathcal{F}'}$	72.84	75.06	72.48	73.77
	$\mathcal{P}_{\beta=4}^{\mathcal{F}'}$	75.04	67.66	76.23	71.94
	$\mathcal{Q}^{\mathcal{F}'}$	75.30	63.27	77.25	70.26
50	$\mathcal{P}^{\mathcal{F}'}$	78.20	47.02	83.25	65.14
	$\mathcal{P}_{\beta=1}^{\mathcal{F}'}$	76.73	77.24	76.65	76.94
	$\mathcal{P}_{\beta=2}^{\mathcal{F}'}$	72.30	81.63	70.79	76.21
	$\mathcal{P}_{\beta=4}^{\mathcal{F}'}$	75.30	74.82	75.38	75.10
	$\mathcal{Q}^{\mathcal{F}'}$	73.76	70.10	74.35	72.22
100	$\mathcal{P}^{\mathcal{F}'}$	78.28	46.85	83.37	65.11
	$\mathcal{P}_{\beta=1}^{\mathcal{F}'}$	81.01	79.21	81.30	80.25
	$\mathcal{P}_{\beta=2}^{\mathcal{F}'}$	80.04	78.23	80.33	79.28
	$\mathcal{P}_{\beta=4}^{\mathcal{F}'}$	78.64	79.20	78.55	78.87
	$\mathcal{Q}^{\mathcal{F}'}$	76.59	74.66	76.91	75.78
200	$\mathcal{P}^{\mathcal{F}'}$	72.19	56.03	74.81	65.42
	$\mathcal{P}_{\beta=1}^{\mathcal{F}'}$	85.90	79.83	86.88	83.35
	$\mathcal{P}_{\beta=2}^{\mathcal{F}'}$	84.15	79.88	84.84	82.36
	$\mathcal{P}_{\beta=4}^{\mathcal{F}'}$	81.51	80.72	81.64	81.18
	$\mathcal{Q}^{\mathcal{F}'}$	81.42	77.04	82.14	79.59
300	$\mathcal{P}^{\mathcal{F}'}$	76.42	51.23	80.51	65.87
	$\mathcal{P}_{\beta=1}^{\mathcal{F}'}$	86.98	80.55	88.02	84.28
	$\mathcal{P}_{\beta=2}^{\mathcal{F}'}$	86.70	79.14	87.93	83.53
	$\mathcal{P}_{\beta=4}^{\mathcal{F}'}$	84.20	81.24	84.68	82.96
	$\mathcal{Q}^{\mathcal{F}'}$	82.73	79.65	83.23	81.44

ACKNOWLEDGMENT

This research is supported by Natural Science and Engineering Research Council of Canada (NSERC). Additionally, the first author gratefully acknowledges the supports from the Killam Trusts, Mitacs, and the province of Nova Scotia. The research is conducted as part of the Dalhousie NIMS Lab at: https://projects.cs.dal.ca/projectx/.

REFERENCES

[1] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys Tutorials*, vol. 18, no. 2, Secondquarter 2016.

[2] S. Dua and X. Du, *Data Mining and Machine Learning in Cybersecurity*, 1st ed. Boston, MA, USA: Auerbach Publications, 2011.

[3] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Comput. Surv.*, vol. 50, no. 3, pp. 41:1–41:40, Jun. 2017.

[4] F. Haddadi and A. N. Zincir-Heywood, "Botnet detection system analysis on the effect of botnet evolution and feature representation," in *Genetic and Evolutionary Computation Conference Companion*, 2015.

[5] D. Zhao, I. Traore, B. Sayed, W. Lu, S. Saad, A. Ghorbani, and D. Garant, "Botnet detection based on traffic behavior analysis and flow intervals," *Computers & Security*, vol. 39, pp. 2–16, 2013.

[6] P. Torres, C. Catania, S. Garcia, and C. G. Garino, "An analysis of recurrent neural networks for botnet detection behavior," in *2016 IEEE Biennial Congress of Argentina (ARGENCON)*, June 2016, pp. 1–6.

[7] A. Saied, R. E. Overill, and T. Radzik, "Detection of known and unknown ddos attacks using artificial neural networks," *Neurocomputing*, vol. 172, pp. 385 – 393, 2016.

[8] D. C. Le, A. N. Zincir-Heywood, and M. I. Heywood, "Unsupervised monitoring of network and service behaviour using self organizing maps," *Journal of Cyber Security and Mobility*, vol. 8, no. 2, 2018.

[9] R. Perdisci, G. Gu, and W. Lee, "Using an ensemble of one-class SVM classifiers to harden payload-based anomaly detection systems," in *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM)*, 2006, pp. 488–498.

[10] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: Methods, systems and tools," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 303–336, 2014.

[11] D. Song, M. I. Heywood, and A. N. Zincir-Heywood, "Training genetic programming on half a million patterns: an example from anomaly detection," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 225–239, June 2005.

[12] F. Haddadi, D. Runkel, A. N. Zincir-Heywood, and M. I. Heywood, "On botnet behaviour analysis using gp and c4.5," in *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*. ACM, 2014, pp. 1253–1260.

[13] S. Khanchi, A. Vahdat, M. I. Heywood, and A. N. Zincir-Heywood, "On botnet detection with genetic programming under streaming data label budgets and class imbalance," *Swarm and Evolutionary Computation*, vol. 39, 2018.

[14] D. C. Le, S. Khanchi, A. N. Zincir-Heywood, and M. I. Heywood, "Benchmarking evolutionary computation approaches to insider threat detection," in *Genetic and Evolutionary Computation Conference*, 2018.

[15] S. Sen and J. A. Clark, "Evolutionary computation techniques for intrusion detection in mobile ad hoc networks," *Computer Networks*, vol. 55, no. 15, pp. 3441 – 3457, 2011.

[16] S. X. Wu and W. Banzhaf, "The use of computational intelligence in intrusion detection systems: A review," *Applied Soft Computing*, vol. 10, no. 1, pp. 1 – 35, 2010.

[17] M. Masud, J. Gao, L. Khan, J. Han, and B. M. Thuraisingham, "Classification and novel class detection in concept-drifting data streams under time constraints," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 6, June 2011.

[18] E. Manzoor, H. Lamba, and L. Akoglu, "xstream: Outlier detection in feature-evolving data streams," in *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

[19] F. Haddadi and A. Zincir-Heywood, "Benchmarking the effect of flow exporters and protocol filters on botnet traffic classification," *IEEE Systems Journal*, vol. 10, no. 4, pp. 1390–1401, 2016.

[20] M. Brameier and W. Banzhaf, *Linear Genetic Programming*, ser. Genetic and Evolutionary Computation. Boston, MA, USA: Springer US, 2007.

[21] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Computers & Security*, vol. 45, 2014.