SAND2011-3273 C

SAND2011-3273C

Adaptive Rule-Based Malware Detection Employing Learning Classifier Systems: A Proof of Concept

Jonathan J. Blount, Daniel R. Tauritz
Natural Computation Laboratory
Department of Computer Science
Missouri University of Science and Technology
Rolla, Missouri, U.S.A.
{*blountj, dtauritz*}*@acm.org*

Samuel A. Mulder
Sandia National Laboratories
Albuquerque, New Mexico, U.S.A.
samulde@sandia.gov

***Abstract*—Efficient and accurate malware detection is increasingly becoming a necessity for society to operate. Existing malware detection systems have excellent performance in identifying known malware for which signatures are available, but poor performance in anomaly detection for zero day exploits for which signatures have not yet been made available or targeted attacks against a specific entity. The primary goal of this paper is to provide evidence for the potential of learning classifier systems to improve the accuracy of malware detection. A proof of concept is presented for adaptive rule-based malware detection employing learning classifier systems, which combines a rule-based expert system with evolutionary algorithm based reinforcement learning, thus creating a self-training adaptive malware detection system which dynamically evolves detection rules. Experimental results are presented which demonstrate the system's ability to learn effective rules from repeated presentations of a tagged training set and show the degree of generalization achieved on an independent test set.**

***Keywords*-Learning Classifier Systems; Malware Detection**

I. Introduction

Malware is an ever-growing threat to computer systems, and security researchers are in a virtual arms race with malware authors. The number of different strains and types of malicious software has been on the rise for years. A recent report states that "in 2010, cyber-criminals created and distributed a third of all existing viruses" [1]. The average number of threats created per day has gone from 55,000 in 2009 to 63,000 in 2010 [1].

The classification of malware is a difficult problem. Software that allows unauthorized control of a system is obviously malicious, but software that displays ads (adware) is not strictly harmful, unless it invades privacy and collects personal information without user consent (spyware). Software can be considered malicious depending on the intent of its author, and this makes it difficult to classify a piece of code as malicious or not. Research has automated malware detection based on the intent of the user and the intent of the malware author [2].

Malware detection is the primary step in preventing a computer system from potential information loss and system compromise. There are a variety of ways to detect malicious software. A majority of anti-virus software use signature-based techniques that utilize a pre-defined set of signatures [3]. This is a reactive approach: until a signature is created for an exploit, the exploit will elude detection by traditional anti-virus software.

The goal of this research is to create a self-training adaptive malware detection system which dynamically evolves detection rules. A learning classifier system (LCS) is a rule-based, expert system with Evolutionary Algorithm (EA) based Reinforcement Learning (RL) [4]. An EA is a population based search technique which uses biological inspired mechanisms (natural selection, mutation, recombination) to evolve solutions to a problem [4]. The LCS uses RL to adjust the fitness of each rule based on environmental reward. The EA evolves the set of rules, replacing some of the weaker rules with newly created rules.

A malware detection system's primary purpose is to have a high rate of detection while maintaining a low rate of false positives. To test a malware detection system, known pieces of malware are presented to it, as well as files that are not malicious (goodware). The set of software used for testing must already be tagged in order to assess how well the system performs. The testing framework for this malware detection learning classifier system uses known sets of malicious and clean files as indicated by VirusTotal[1].

The remaining sections are organized as follows: Section II provides background on LCS and malware detection techniques, Section III details the methodology of the detection system, Section IV explains the experimental setup, Section V presents preliminary results, Section VI discusses future directions, and Section VII summarizes the research.

II. Background

John Holland created the precursor to the LCS around his Genetic Algorithm, which later became the LCS when it included an RL component [4]–[6]. The basic framework of

[1]VirusTotal - http://www.virustotal.com

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

an LCS consists of (1) a finite population of classifiers that represents the current knowledge of the system, (2) a performance component, which regulates interaction between the environment and classifier population, (3) a reinforcement component, which distributes the reward received from the environment to the classifiers and is the learning mechanism, and (4) a discovery component which employs an EA to evolve better rules and improve existing ones [4]. RL serves two purposes: (1) to promote individuals that obtain high rewards and (2) to discover better rules [4].

LCSs can be applied to different problem domains including optimization problems, classification problems, and RL problems [7]. Optimization problems are solved by searching a solution space for the best solution. In a classification problem, the LCS learns to which class each problem instance belongs. Feedback is immediate and problem instances can be sampled independently. Contrary to classification problems, feedback in RL problems provides an indication of the quality of an action and may not be immediate. Classification problems can be redefined as single-step RL problems where reward is immediate. Malware detection is a boolean classification problem, where each problem instance (a file) can be classified as one of two classes (malicious or non-malicious). This classification problem can be learned by an LCS when converted into a single-step RL problem. The learning system is desired to have a high percentage of correct classifications (accuracy) and able to classify unseen problem instances (generality) [7].

Recent research has shown that existing anti-virus software is poor at identifying polymorphic scripts [8]. By using various polymorphic techniques, the authors were able to create malicious scripts with identical functionality to known malware that were undetectable by anti-virus software. They were able to detect the malware variants by analyzing the software's dependency graph with a hybrid genetic algorithm. Even simple polymorphism could fool more than half of the software that detected the original malware.

There have been approaches to malware detection that use nonsignature based techniques. By extracting features of portable executable (PE) files, malicious executables can be detected by heuristic techniques [3]. Packed and nonpacked files were processed separately by a decision tree to determine if they were malicious or not. This technique was shown to overcome a bias shown by structural features for packed/nonpacked executables.

A comparative study analyzed evolutionary and nonevolutionary rule learning algorithms to evaluate performance differences [9]. Five types of LCS were compared with five nonevolutionary rule learners. Their conclusions indicated that the nonevolutionary rule learning algorithms outperformed the LCS types. The LCS types still had very high detection rates, but at the expense of high processing time. The authors acknowledged that they did not explore different

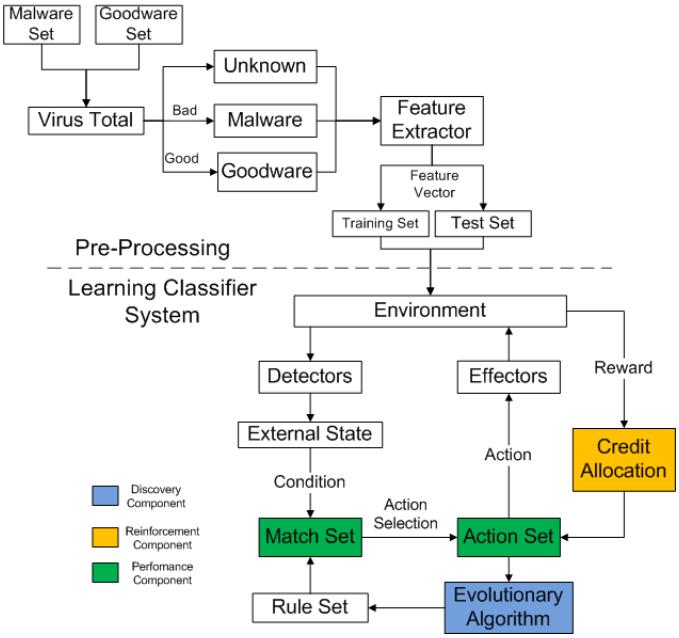

Figure 1. Malware LCS Diagram

configuration parameters for the rule learning algorithms, as well as a plan to combine the datasets into a single large set in order to create a more challenging environment. Limiting the sets to a specific category of malware increases the chances of similarity between files, providing for an easier environment than a random collection of malware.

The research presented in this paper demonstrates promising results for a custom built malware detection system employing LCS on a dataset representative of the real world. The test and training sets were not limited to a specific category of malware; they contained a collection of malware seen in the real world. Another step this research took is to make sure the files in the sets were actually considered malicious or clean by submitting them to VirusTotal. Not all samples from the malicious collection were known to be malicious.

III. Methodology

The LCS environment is a collection of files, both malicious and non-malicious. Malicious software samples were obtained from Offensive Computing[2]. Non-malicious software samples were obtained from a machine freshly installed with Windows XP and a university campus computer learning center machine running Windows XP. Clean executables included software created by Microsoft, Adobe, MathWorks, other third parties, and open source software. The malware detection system consists of a pre-processing component and an LCS; an overview of the process is presented in Fig. 1.

[2]Offensive Computing - http://www.offensivecomputing.net/

A. Pre-processing

The pre-processing stage analyzes all of the sample files and submits them to VirusTotal, which tests files against 43 anti-virus engines, for a determination if a file is considered malicious or not. The set of samples from Offensive Computing does not contain just malware. If more than 25 percent of the anti-virus vendors VirusTotal uses to scan a file report malicious, the file is considered malicious. This eliminates the possibility of a single anti-virus software misclassifying a file. Each file was checked to make sure that VirusTotal is consistent with the dataset the file came from; i.e., a sample from the malware set must be identified as malicious and a sample from the goodware set as non-malicious. If it is not consistent, the sample is considered unknown. Furthermore, all samples (malicious or not) that are not in VirusTotal's database are also considered unknown. All unknown samples are not used in the system, as they can not be verified as being definitely malware or goodware.

The executable datasets consist of software that runs on the Windows operating system. Windows executables are written in the PE format. PE files contain a number of sections, each of which has a header that describes its data and resources. One important section is the import data section which contains the Import Address Table (IAT). The IAT is where every external function called by an executable is stored. This table includes the name of the function and the name of the dynamic link library (DLL) in which the function is stored. This research assumes that malicious files will be distinguishable from goodware based upon the structure of the PE file including the table of imported functions. While in the real world this assumption would not always hold, for the purpose of this research, this is an acceptable assumption for determining whether an LCS can potentially be used for malware detection. The IAT from each file is used to generate a feature list containing all of the imported functions each executable references. The feature extraction part of the system was implemented using an open source tool called pefile[3].

B. Learning Classifier System

The LCS used is based on the eXtended Classifier System (XCS) [10]. XCS uses an accuracy based fitness, an EA that acts on the action set, and a Q-Learning RL technique [4]. Previous strength-based LCSs used a strength value for both fitness in reproduction selection and as a measure of reward the rule receives, which is used for action selection. The accuracy-based XCS introduced three new components to maintain separate estimates used for reproduction and action selection: *reward prediction* estimates the average reward a rule receives when its condition is met, *prediction error* estimates the deviation of the prediction, and *fitness* estimates a scaled relative accuracy of the rule [11]. In XCS, rules with consistent strengths will have accurate predictions, and rules with varying strengths will have inaccurate predictions [12]. Each rule's fitness is derived from the estimated accuracy of reward predictions instead of from the reward predictions themselves [7]. XCS is designed to evolve a representation of the best solution for all possible problem instances and a complete and accurate payoff map of all possible solutions for all possible problem instances [7].

[3]pefile - http://www.code.google.com/p/pefile/

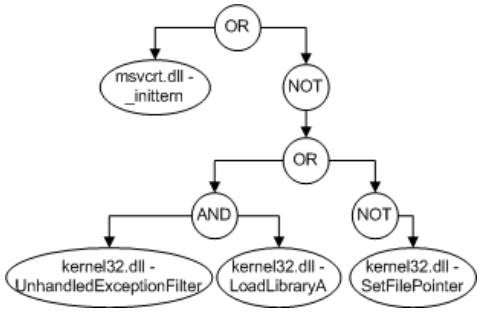

Figure 2. Visualization of a subtree of a generated rule

The EA operates at the level of individual rules, known as a Michigan-style LCS. The entire population of rules represents a solution. A Pittsburgh-style LCS represents a population of variable length rule sets [13]. Each rule set is a solution, and the EA operates on the level of an entire rule set [4]. Each style of LCS is best applied to a certain type of learning. Pittsburgh style LCSs are usually applied in "offline" or batch learning scenarios, where all training problems are presented simultaneously to the learner which results in a rule set that does not change over time [4], [7]. Michigan style LCSs are designed to work "online", incrementally learning each problem instance individually and evolving the rule set over time with each new observation. Offline learning is characteristic of data mining problems, and this research uses a Michigan style LCS to continuously evolve a malware classifier.

1) Population: After pre-processing all of the files in the dataset, the LCS randomly initializes a population of rules using the extracted features. A traditional LCS encodes the condition as a bit-string, but in this system each condition is represented as an s-expression. S-expressions can be visualized as a tree structure where internal nodes are one of the logic operators $\{AND, OR, NOT\}$ and leaf nodes (terminals) are a single feature (an example is shown in Fig. 2). Since malware detection is a binary classification problem, the action represents whether the file is classified as malicious or not. If no rules match a given malicious file during training, a covering operator creates a rule that has a matching condition and inserts it into the population.

2) Action Selection: A single problem instance in this system is a randomly chosen file that is presented to the system. Each rule in the population is a parse tree, and is compared to the extracted features from the file. If the parse

tree matches the feature, it is put into the LCS's match set. From the match set, an action is chosen.

There are various methods of choosing an action, and XCS typically alternates between two methods in an explore/exploit scheme [4]. Explore randomly selects an action from within the action set, while exploit deterministically selects the action with the highest reward. Once an action is chosen, the action set becomes the rules in the match set that advocate the chosen action.

3) Rule Evaluation: The rules in the action set are updated every iteration with reward from the environment. In the classification problem of malware detection, environment feedback is immediate and based on whether the action selection was correct. In XCS, the RL technique is based on the Q-learning algorithm [14]. Three parameters are adjusted to determine the performance of a rule, in the order: prediction error, prediction, fitness [7].

A feature is considered malicious or benign based on whether the file it was extracted from was identified as malicious or benign. It is important to realize that not all features of a malicious file are themselves malicious; many malicious files have benign actions and there is an overlap between the sets of benign features and malicious features. Reward is based on whether the chosen action matches the classification of the file.

4) Mutation: Mutation introduces random variation to individuals by selecting a random mutation point. The subtree rooted at that node is replaced with a randomly generated tree. The height of the generated subtree is limited so the entire tree does not exceed the $maxheight$ parameter, in order to keep processing time from growing indefinitely. The terminal nodes were chosen using proportionate selection from the vector of features extracted from the training set. Features were chosen using roulette-wheel selection, with a probability proportional to the number of times a feature appeared in the training set.

5) Crossover: Crossover works similar to that in Genetic Programming. A subtree starting at a random node is chosen from each parent, and the subtrees are swapped to create two new children. During recombination the second parent's random node selection was limited to those nodes which would keep the two created children's heights below the $maxheight$ parameter.

6) Evolutionary Algorithm: In XCS, the EA reproduces rules in the action set, realizing implicit niching [11] as opposed to panmictic reproduction, where rules are selected from the entire population. XCS performs genetic reproduction if the average time since the last EA invocation of the rules in the action set exceeds threshold θ_{GA}.

The mechanism used for parent and survivor selection is tournament selection. Parent selection chooses a set of classifiers at random, and the one with the highest fitness is chosen to become a parent. In XCS an effective method for determining parent tournament size is to make it proportional to action set size [7]. Parameter τ represents the proportion of the action set that is used in the tournament. Since the EA acts on individuals in the action set, selective pressure is based on the size of the action set. If selection pressure is too weak, learning may not take place and if selection pressure is too strong, crossover never has any effect since identical individuals are crossed [7]. Survivor selection repeatedly executes a tournament of a user specified size and deletes the least fit until the population size has been reduced to its specified size. An age requirement has to be met before rules are considered for parent and survival selection; a rule is not eligible for selection until the system has presented it with θ_{age} files.

IV. Experimental Setup

The goal of this system is to evolve rules using RL that will identify malware. The pre-processing step first extracts features from the files and this collection of features is divided into training and testing sets. The LCS evolved rules over the training set, then evaluated them over the testing set. The list of system parameters is presented in Table I.

The feature that was extracted from the files was the list of imports from the IAT. This limited feature was able to produce promising results by itself, and shows the usefulness of an LCS to enhance malware detection. The number of imports varied per file, and there was a noticeable difference between non-malicious and malicious sets. Malicious files generally have a fewer number of imports than non-malicious files. This is logical as the non-malicious set contains general DLL files, which are shared libraries offering a wide range of functionality, and malware is typically written to specifically target one vulnerability.

Occasionally, a malicious file imports an identical set of functions as a non-malicious file. Any detection system would not be able to tell the difference between files with identical features. These files were removed from the dataset to increase the usefulness of the evolved rules. Files with corrupt or missing IATs could not be analyzed in this system. Some malicious executables may have purposefully modified PE headers to make analysis harder. Extracting additional features would improve the number of files that could be used in the system; these features are discussed in the future work section. After removing incompatible files, the experiments were run with 6000 files using a stratified ten-fold cross-validation test. Each fold contained 50% malware and 50% goodware.

A rule's classification accuracy on a file is one of the following four categories:

1) True positive (TP): correctly classifies a malicious executable as malicious.
2) False negative (FN): incorrectly classifies a malicious executable as non-malicious.
3) True negative (TN): correctly classifies a non-malicious executable as non-malicious.

Table I
CLASSIFIER SYSTEM PARAMETERS

Parameter Name	Parameter Value
Initialization	Uniform Random
Population Size (μ)	1000
Operators	['and', 'or', 'not']
Operator Rate	0.75
Max Tree Height	10
Parent Tournament Proportion (τ)	0.2
Survivor Tournament Size	10
Crossover Rate (χ)	1.0
Mutation Rate (μ)	0.04
EA Threshold (θ_{EA})	25
Learning Rate (β)	0.2
Accuracy Determination (α)	0.1
Error Threshold (ϵ_0)	10
Age Threshold (θ_{age})	300

4) False positive (FP): incorrectly classifies a non-malicious executable as malicious.

Three different metrics were tracked: (1) classification accuracy, (2) detection rate, and (3) false alarm rate (FAR). These are defined mathematically as:

$$\text{classification accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

$$\text{detection rate} = \frac{TP}{TP + FN} \quad (2)$$

$$\text{false alarm rate} = \frac{FP}{FP + TN} \quad (3)$$

The classification accuracy rates how the system performs in general, while detection rate and false alarm rate show more specific metrics on the trade-offs between malware coverage and false positives.

V. RESULTS & DISCUSSION

Results were averaged over the ten-fold cross validation experiment. Fig. 3 shows the results from the system's overall accuracy and the average accuracy of the individual rules for both training and testing using the standard XCS selection technique.

Three different parent and survivor selection techniques were compared in order to determine how the selection mechanism affects results. The three methods were strength (classification accuracy), accuracy (traditional XCS rule fitness), and prediction (the metric used for action selection). Individual classification accuracy, averaged over all rules in the population, is shown in Table II and system classification accuracy in Table III.

Selection based on classification accuracy is a strength-based technique. It improves the average individual at the expense of the system. This appears to be due to evolution selecting rules able to classify easy-to-distinguish files, and not selecting rules able to classify difficult to detect malware.

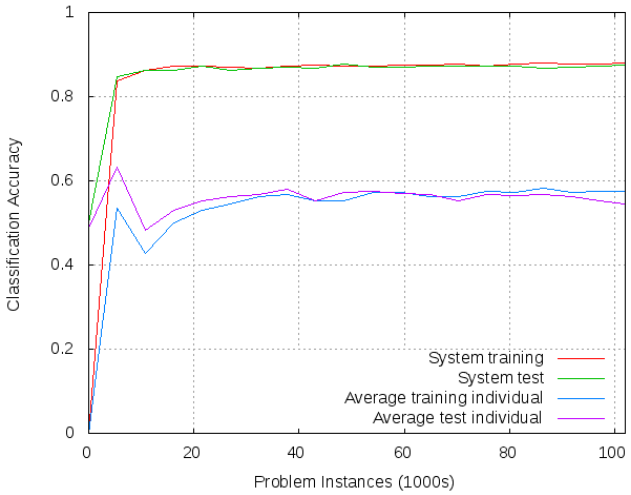

Figure 3. System and Individual Accuracy Rates

Table II
SYSTEM RATES FOR SELECTION METHODS

Selection	System Accuracy	Detection Rate	FAR
Strength	0.768 (0.018)	0.767 (0.035)	0.230 (0.021)
Prediction	0.841 (0.010)	0.824 (0.021)	0.146 (0.013)
Accuracy	0.880 (0.004)	0.870 (0.013)	0.112 (0.011)

Rules compete for the highest individual accuracy, instead of working to improve system accuracy.

Using fitness for selection (traditional XCS) resulted in a lower average rule accuracy but a higher system accuracy. Each rule was consistent in the files it classified, and as the system learned which rules consistently misclassify files, it could still utilize those rules in making accurate predictions.

The third technique, rule prediction, combined the previous two approaches, accuracy and strength. The metric used for action selection, multiplying rule prediction by rule fitness, was also used for selection in the EA. This allowed for a balance between consistent and accurate rules. This technique performed in the middle of the three; the average rule accuracy was closest to the overall accuracy of the system.

Two-sample F-tests for equal variances and corresponding two-tailed t-tests using $\alpha = 0.05$ were run for all results. The p values for all tests were small enough to reject the null hypothesis, indicating that the tests are statistically significant. There is a balance between individual and overall system classification accuracy; using a strength-based selection method improved individual accuracy and using an accuracy-based method improved the system as a whole. The purely accuracy-based method performed the best overall, the hypothesized benefit of combining the advantages of the accuracy technique with the increased competition of the strength technique did not materialize.

Table III
AVERAGE INDIVIDUAL RATES FOR SELECTION METHODS

Selection	Individual Accuracy	Detection Rate	FAR
Strength	0.945 (0.005)	0.961 (0.005)	0.020 (0.004)
Prediction	0.905 (0.013)	0.933 (0.013)	0.031 (0.005)
Accuracy	0.595 (0.032)	0.677 (0.036)	0.318 (0.025)

VI. FUTURE WORK

Future work for this research includes expanding the feature set to include other aspects of PE files. Besides the IAT, other features to investigate include the name and size of all PE sections, the entropy of each section, and other aspects that can be statically extracted. Additional features would allow files with corrupt or missing IATs to be analyzed.

Packed files were not given special consideration in this study, although packed files have different structural features than unpacked files. Future work will determine how these features affect the detectability of malicious files. The current system cannot tell the difference between files that import the same list of functions, and this decreased the total size of the dataset on which the system could run. By expanding the set of features the LCS uses, files can be better distinguished, and detection rules, as well as the overall system, will evolve to become more accurate.

Benchmarking the system on larger, more diverse datasets will provide additional validation of its potential for malware detection. Tuning of the LCS, including the selection technique, may be expected to increase system performance.

VII. CONCLUSION

This paper presents a proof of concept for adaptive rule-based malware detection employing an LCS. It combines a rule-based expert system with EA based RL, creating a self-training adaptive malware detection system which dynamically evolves detection rules. The LCS was extended from XCS to use s-expressions for the rule's condition. Promising initial results are shown; although their accuracy has to be further improved to be competitive with the state-of-the-art. Evidence for the feasibility of using an LCS to evolve rules is provided. With more features, additional aspects of PE files could be analyzed, which would enhance detection rates and lower false alarm rates.

The dataset included malware samples from Offensive Computing and non-malicious samples from Windows computers. The set of known malicious and non-malicious files were processed to confirm their maliciousness and features were extracted to be used for rule conditions. Experimental results demonstrate the system's ability to evolve effective rules based on a training set, and its ability to generalize to previously unseen samples contained in a test set.

ACKNOWLEDGMENT

The authors would like to thank Danny Quist of Offensive Computing for providing the malware samples. This research was funded by Sandia National Laboratories.

REFERENCES

[1] PandaLabs, "Annual Report PandaLabs 2010," Panda Security, Tech. Rep., 2010.

[2] W. Cui, "Automating malware detection by inferring intent," Ph.D. dissertation, University of California at Berkeley, Berkeley, CA, USA, 2006.

[3] M. Shafiq, S. Tabish, and M. Farooq, "PE-Probe: Leveraging Packer Detection and Structural Information to Detect Malicious Portable Executables," in *Virus Bulletin Conference (VB), Switzerland*, 2009.

[4] R. J. Urbanowicz and J. H. Moore, "Learning classifier systems: a complete introduction, review, and roadmap," *J. Artif. Evol. App.*, vol. 2009, pp. 1:1–1:25, January 2009.

[5] J. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. The MIT Press, 1975.

[6] L. Booker, D. Goldberg, and J. Holland, "Classifier systems and genetic algorithms," *Artificial intelligence*, vol. 40, no. 1-3, pp. 235–282, 1989.

[7] M. Butz, *Rule-based evolutionary online learning systems: A principled approach to LCS analysis and design*. Springer Verlag, 2006.

[8] K. Kim and B.-R. Moon, "Malware detection based on dependency graph using hybrid genetic algorithm," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, ser. GECCO '10. New York, NY, USA: ACM, 2010, pp. 1211–1218.

[9] M. Shafiq, S. Tabish, and M. Farooq, "On the appropriateness of evolutionary rule learning algorithms for malware detection," in *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*. ACM, 2009, pp. 2609–2616.

[10] S. Wilson, "Classifier fitness based on accuracy," *Evolutionary computation*, vol. 3, no. 2, pp. 149–175, 1995.

[11] M. Butz, T. Kovacs, P. Lanzi, and S. Wilson, "How XCS evolves accurate classifiers," in *Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001)*, 2001, pp. 927–934.

[12] T. Kovacs, *Strength or accuracy: credit assignment in learning classifier systems*, ser. Distinguished dissertation series. Springer, 2004.

[13] S. F. Smith, "A learning system based on genetic adaptive algorithms," Ph.D. dissertation, University of Pittsburgh, Pittsburgh, PA, USA, 1980.

[14] C. J. C. H. Watkins, "Learning from Delayed Rewards," Ph.D. dissertation, King's College, Cambridge, 1989.