Submitted to the 2017 International Conference on Cloud and Autonomic Computing

A Self-Protection Agent using Error Correcting Output Codes to Secure Computers and Applications

Fabian de la Peña Montero, Salim Hariri and Gregory Ditzler

Abstract— **The human immune system is incredibly efficient at identifying self- and non-self entities in our bodies. A non-self entity (malicious), once identified, is attacked by certain types of cells to remove the intruder before it can cause damage. Our immune system has components that not only identify non-self entities, but also recall old entities that may not have been encountered for a very long time, but it is still very important that these entities be correctly classified as malicious. The domain of cybersecurity can significantly benefit from having a framework that is able to identify, react and adapt to malicious behaviors. Such a model for cyber protection should draw a parallel to our immune system, at least at a high level. In this work, we present a flexible framework that leverages machine learning to identify malicious behaviors that are threats to users, computers and applications in a network. We have identified this framework after considering the components that are required to allow the system to identify, react and adapt to malicious behaviors. The proposed framework relies on the collection and aggregation of information relevant to identifying such malicious behaviors, machine learning – to automatically learn and identify non-self behaviors, and an adaptation mechanism to incorporate new threats for future classification. We benchmarked the proposed approach on a data set collected from multiple users, computer and applications, and we show that attacks (i.e., non-self behaviors) can be identified and mitigated through software. We compared classification models that perform binary classification (i.e., self or non-self), as well as multi-class predictions (i.e., what type of non-self behavior is being detected).**

I. Introduction

There is an increasing dependency of individual users, as well as businesses, on secure computers and computer networks. Furthermore, there is also an increase in the threats that an adversary can compromise them by using malicious agents. Thus, there is an urgent need for the accurate, adaptive and automatic detection of cyber-attacks and intrusions has emerged. The ever increasing need for the adaptive, autonomic detection of and protection against attacks/intrusions is evident given the nature of today's technological climate. Current cyber-security approaches are failing to stop hackers from infiltrating cyber-systems, which is apparent given the recent breaches in security. These intrusions have been occurring with ever increasing frequency in the last decade. For example, in September of 2016, Yahoo revealed that in 2014 it had an intrusion that resulted in the theft of at least 500 million Yahoo accounts [1]. In another instance, Quest Diagnostics, a New Jersey-based medical laboratory company, had a data breach on Monday December 12, 2016 in which the personal and medical data of about 34,000 people was stolen [2]. In yet another instance, which occurred on May, 2016, the names and passwords of more than 360 million Myspace accounts were stolen [3]. These are just some of the numerous intrusions that have occurred or have been reported in this year alone. Also, one needs to keep in mind that these samples are for high profile cases that get the attention of the media. For each case that is reported there are many others that occur without public notice.

The need for intelligent and autonomic methods for identifying malicious types of behaviors cannot be overstated enough. In autonomic management, the cyber infrastructure and services would be seamlessly managed (self-management) with little involvement by users and system administrators [4]–[6]. In this paradigm, the management system seamlessly adopts its policies and control algorithms to meet their performance requirements (self-optimize), reconfigure its resources to tolerate hardware and/or software faults (self-heal), and stop and/or mitigate the impacts of threats and cyber-attacks (self-protect).

These types of behaviors from an autonomic management system for malicious threat detection has strong parallels to that of the human immune system. At a high level there are components to continuously monitor and observe a system for non-self entities [7]–[9]. Once a non-self entity (abnormal/malicious) is detected, the threat is removed from the system. The non-self entity is added into the memory (i.e., learned from) of the "cyber" immune system to detect such an entity at a future time. A cyber-defense system should leverage the same types of qualities, namely: continuous surveillance, threat mitigation and autonomic learning.

In this contribution, we introduce a new framework that is inspired by the high-level components of the human immune system to protect users and applications from malicious cyber activities. The proposed *Self-Protection Agent* (SPA) models behaviors of users and adversaries then uses machine learning to provide a predictive model that can be deployed in a real-time environment thatontinuously monitor users and applications. We demonstrate the effectiveness of the proposed approach by benchmarking the SPA real-world data, and we apply multiple types of attacks to normal operation of computers. We observe very high classification rates of non-self (malicious) activities. Furthermore, we implemented the SPA in a real system and confirm the experiment that were performed offline during the SPA validation phase.

This manuscript is organized as follows: Section II dis-

F. de la Peña Montero, G. Ditzler and S. Hariri are with the Department of Electrical & Computer Engineering at the University of Arizona, Tucson, AZ 85721. FPM and SH are also affiliated with the Cloud and Autonomic Computing Center and GD is affiliated with the Cognitive Sensing Research Center at the University of Arizona.

Email: {fdlpm, hariri, ditzler}@email.arizona.edu

cusses related work in cyber self-protection, Section III provides a detailed discussion of the SPA from the data collection to threat mitigation, Section IV presents the preliminary results of the SPA evaluated on threats to users and applications, and Section V draws conclusions from this work.

II. Related Works

A. Cyber Threat Identification

There is a significant effort to address cyber threat identification in response to a failure of keeping computers, users and applications safe from attacks. Khorshed *et al.* proposed a system where insider activities in a cloud system are monitored and modeled through machine learning. Their approach used a C4.5 decision tree in Weka to predict on data [10], [11]. The goal of the system was to be able to classify the activities into the following categories: reboot physical machine, malicious insider cloning virtual machine (VM), malicious insider copying everything from VM, malicious insiders taking snapshot of VM, installing a new guest VM on the same physical hardware, and turning on any guest VM on the same physical hardware [12]. This approach for threat identification is similar to our own in the fact it also uses machine learning; however, they focused on the specific case of insider threats in a cloud system whereas our framework can be applied to any cyber-infrastructure. The work in [13] proposed performing data mining on log files to detect patterns that allow for the detection of brute force password cracking and denial of service attacks. This method relies on system logs in order to function and is ineffective against malicious activities which effectively hide and do not show on logs. Our behavior based approach is able to detect these threats without reliance on logs.

Forrest *et al.* proposes a methodology in which computer systems are protected by distinguishing self from other. This is done through a change detection method based on the generation of T cells in the immune system [14]. The system proposed only focuses on the detection of computer viruses and does not, at the time of the paper's writing, take the entire infrastructure into account like our approach.

Our approach builds upon the previous research performed at the NSF Cloud and Autonomic Computing Center (CAC). The framework presented in this paper is an extension of the autonomic computing paradigm developed at CAC. The autonomic computing paradigm was inspired by the human autonomic nervous system. Under this paradigm, an autonomic computing system should be a system that has self-awareness, is self-protecting, self-optimizing, self-healing, self-configuring, contextually aware, open and anticipatory [4]. In addition to all these criteria, the proposed framework adds another dimension based on the human immune system by using machine learning to model the behavior of users, computers and applications and then using these models to detect cyber attacks.

The behavior analysis modeling as well as the autonomic computing paradigm used in the proposed framework were previously implemented as part of the Multi-Level Anomaly Based Autonomic Intrusion Detection System [15]. This system provides defense against both known and unknown network attacks. It makes use of autonomic computing to automate the system and defends the network by detecting abnormalities in network operation that deviate from normal network behavior. The framework presented in this work extends upon this idea by also protecting computers, users and applications against cyber attacks.

III. A Framework for Threat Detections With Ensembles

A. An Overview of the Proposed Approach

Figure 1 shows an overview of our proposed pipeline for a self-protection agent (SPA). We briefly discuss the general framework at a high level before going into more detail about each component. In ordered to build a SPA there needs to be data that is valuable for protecting users, computer and applications. Furthermore, the data must be informative to the entity being protected and the data must be able to characterize each entity uniquely. The uniqueness of the entity that are represented by the data collection is analyzed using exploratory data analysis to identify the types of machine learning model that should be used in the SPA. The output of the exploratory data analysis phase is used to learn and validate a model for SPA. Finally, the SPA is deployed in a real-time system with system intervention if malicious behavior is classified.

While this framework is somewhat generic, it is designed so that at each phase the SPA is using the best view of the information that is available to be successful in practice.

B. Data Collection and Exploratory Data Analysis

A software agent was created for Windows computers for the data collection and aggregation. The Windows software agent was written using C#. C# was chosen as the language for this agent since it allows for the easy gathering of information about the computer. This programming language has many built in functions that provides information about computers, users as well as applications. A large amount of data and information is obtained by using C# to perform queries to the Windows Management Instrumentation (WMI) API. WMI is a core Windows management technology that can be used to manage both local and remote computers. WMI can collect information about the internal state of computer systems and it models objects such as disks, processes using classes. The software agent is meant to run continuously in the background in a loop, gathering the latest values of the collected features and inserting them as individual records associated with a time stamp in the central database during each loop iteration.

From the database, the data can later be extracted as CSV files for processing and analysis. The database is composed of the tables shown in Figure 2.

The systems table stores a record of each computer that is monitored and protected by the SPA. The users table stores a record of each user that is monitored and protected by the SPA. The computer_records table stores the data collected

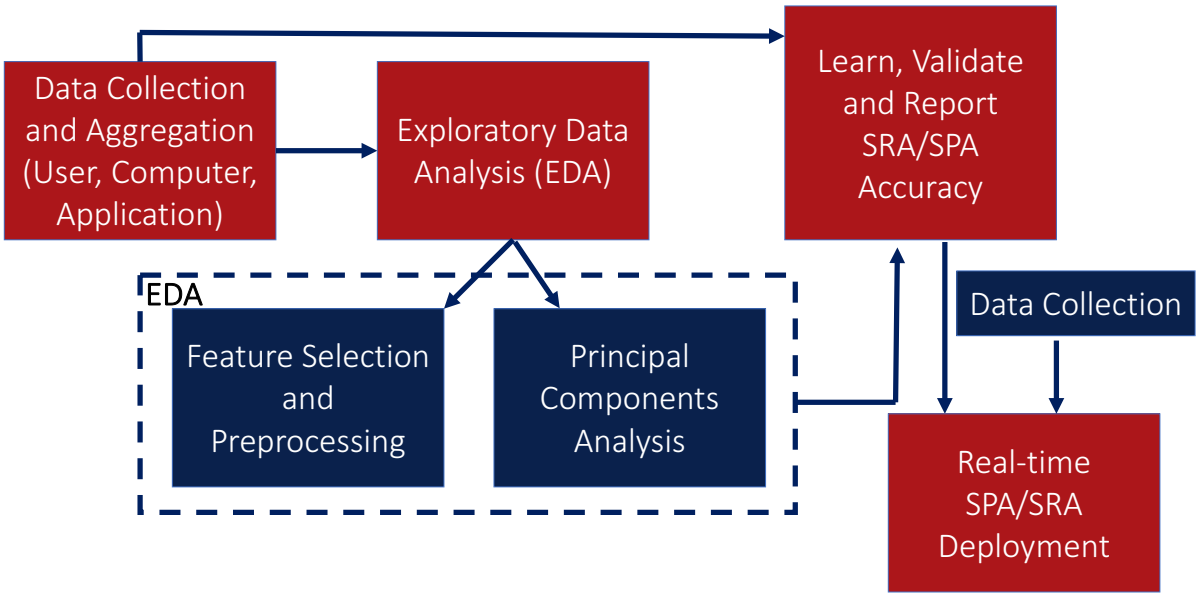


Fig. 1. Overview of the data science pipeline for learning and validating a self-protection agent (SPA). The SPA is designed to protect users/computer and applications from various types of malicious behaviors. The framework follows a methodical processes to select the appropriate models for the SPA.

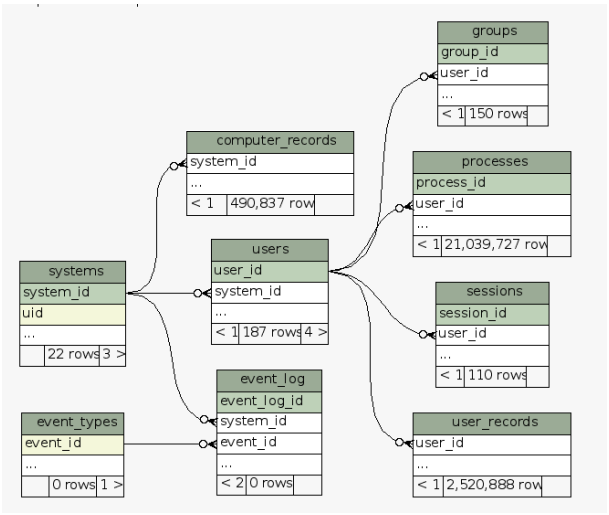


Fig. 2. Self Recognition Database Schema

for all computers monitored by the SPA. The user_records table is analogous to the computer_records table but for user data. The processes table is analogous to the previous two tables but for application/process data.

The Self Recognition Database stores the following features for computers, users and applications (see Appendix). These features were chosen for several reasons. First, boolean features such as Firewall Active, Anti-virus Enabled and Anti-virus Updated were chosen because they can by themselves already let us know if the computer is not in an ideal state. Any change in these type of features from 0 to 1 is a strong indicator of malicious actions. Second, numeric features such as CPU Load, Memory Load, TCP Listener Connections and Active TCP Connections do not by themselves necessarily indicate that there are non-self/malicious activities happening at a computer or being performed by an application or user, but by creating a model that takes all of these features into account, we can detect a change from the norm and thus detect non-self and potentially malicious behavior. We collected and gathered as many features as we could obtain from WMI in order to attempt to completely characterize each computer, user and application with the knowledge that features that ended up not actually contributing to the characterization would be dropped during feature selection at the data analysis phase.

We perform exploratory data analysis on the features collected into the SPA database (see Figure 1). Our first step is to gain an understanding about the separation between the normal and malicious profiles as understanding this allows us to intelligently select the appropriate model for classification. The primary reason for performing this analysis is Occam's razor [16], [17], which states that we should select the simplest model that allows the SPA to model the data. Figure 3 shows a feature by feature scatter matrix plot. In this figure, each numeric feature in the data set is plotted w.r.t. all of the other feature (i.e., the third entry in the first row plot the feature X_1 on the x-axis and feature X_3 on the y-axis). The diagonals show the histogram of a single feature separated by class. The red and black points represent normal (self) and malicious (non-self), respectively. Note that the data are standardized and scaled prior to these plots. There are several observations we can make from these plots:

- The joint informativeness of many of the pairwise comparisons is quite low and there are not two sets of features that can be easily identified to easily separate the data.
- Many of the scatter plots show regions of linear correlation between many of the data samples; however, due to the overlap between the normal and malicious

samples these linear relationships.

In an attempt to get a better understanding of the data, we apply principal components analysis (PCA) to the data collected from users/computers to visualize the data in 3D [16], [17]. The objective for PCA, which finds a projection that maximizes the variation is given by:

$$\text{Var}(Z) = \max_{\mathbf{w}\in\mathbb{R}^p} \left\{ \frac{1}{n-1}\mathbf{w}^\mathsf{T}\mathbf{X}\mathbf{X}^\mathsf{T}\mathbf{w} \right\}, \text{ s.t. } \|\mathbf{w}\|_2^2 = 1.$$

where $\mathbf{w}$ projects the original variable $\mathbf{x}$ onto a line; however, $\mathbf{w}$'s projection is the one that maximizes the variance such that $\mathbf{w}$ is a unit vector. These projection vectors are the eigenvectors of the covariance matrix of data matrix $\mathbf{X}$, where the eigenvectors corresponding to the largest eigenvalues are the projections that contains the most variation. Figure 4 shows the projection of user data onto a 3-dimensional space and the cumulative percentage of variance contained in the leading eigenvectors. Note that we have standardize the data prior to performing the projection. There are a few key observations to take away from these plots. Figure 4(a) shows a clear separation between the normal and malicious profiles, but we also observe that the different types of malicious samples cluster together. It is also important to not (while difficult to see from Figure 4(a)) that there is some overlap between the classes. Secondly, most of the variation, roughly 80%, is contained in the first three principal axis. Thus, the complexity of the classifier need not be large based on the visualization.

C. Threat Identification with an Ensemble of Support Vector Machines

As shown in Figure 1, the SPA takes the data from collectors running on protected machines, which are stored in the Self Recognition Database (SRD), to build a classification module that is designed to protect computers and applications. To achieve this objective, we use an ensemble of classifiers to identify malicious behaviors for the SPA. An ensemble of classifiers is when multiple classifiers are trained on a task (i.e., data set) then at testing time, their outputs are strategically combined [18]. An ensemble-based approach was selected for the SPA based in several reasons that can be summarized as: (a) ensembles have been shown to perform empirically well on a broad set of problems [19]; (b) ensembles have some nice theoretical properties (e.g., Boosting ensembles [20] and the Condorcet's Jury theorem); (c) ensembles work well on problems when there is too much or little data [21], [22]; and (d) ensembles have the ability to exploit the power of weak learnability.

The support vector machine was down selected as the ensemble's base classifier [23], [24], and their popularity was one reason for their selection [25]–[28] Furthermore, the support vector machine solves binary problems, which is a natural fit do discriminating between self- and non-self patterns, The support vector machine is a classifier that maximizes the margin between two classes (e.g., malicious or normal) [23]. Given a data sample $\mathbf{x}$, we need to determine the class (i.e., self- or non-self). The support vector machine makes a prediction $\widehat{y} = \text{sign}(\mathbf{w}^\mathsf{T}\Phi(\mathbf{x}) + b)$. If $\widehat{y} = +1$ than $\mathbf{x}$ is normal otherwise $\widehat{y} = -1$ is a malicious sample. $\Phi(\cdot)$ is a non-linear projection to a higher dimensional space, which allows us to solve non-linear classification problems and $\mathbf{w}$ is the vector that the support vector machine optimizes. Though the primal form of the support vector machine is easiest to interpret, we present the commonly used dual form of the optimization problem, which is given by:

$$\arg\max_{\alpha\in\mathbb{R}^n} \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j K(\mathbf{x}_i,\mathbf{x}_j)$$
$$\text{s.t.} \sum_{i=1}^{n}\alpha_i y_i = 0,\ 0 \le \alpha_i \le C$$

where C is a box constraint for the width of the margin, α_i are Lagrange multipliers and $K(\mathbf{x}_i,\mathbf{x}_j)$ is given by

$$\Phi(\mathbf{x}_i)^\mathsf{T}\Phi(\mathbf{x}_j) = K(\mathbf{x}_i,\mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|_2^2}{\sigma}\right)$$

which is a radial basis function kernel. This kernel allows the support vector machine to solve non-linear classification problems and avoid the calculation of $\Phi(\cdot)$ explicitly by using the kernel trick. The classification function is given by:

$$\begin{aligned}\widehat{y} &= \text{sign}\left(\left(\sum_{i=1}^{n}\alpha_i y_i \Phi(\mathbf{x}_i)\right)^\mathsf{T}\Phi(\mathbf{x}) + b\right)\\ &= \text{sign}\left(\sum_{i=1}^{n}\alpha_i y_i\ K(\mathbf{x}_i,\mathbf{x}) + b\right)\end{aligned}$$

where we have used the definition of $\mathbf{w}$ (see [29]). The free parameters for the support vector machine are C and σ, which are both bound lower bounded by zero. Luckily, the optimization task shown above is a quadratic programming problem that can be efficiently solved using sequential minimal optimization [30].

One of the important aspects of our SPA is the ability to learn and adapt to multiple types of attacks when they are made available; however, we would prefer to avoid completely retraining from scratch. Therefore, we can use an incremental implementation of the support vector machine [31], or an incremental version of the base-classifier that a user would choose for their SPA. An SVM, even the incremental one presented in [31], are binary classifier that can only predict on two classes. While this is useful for identifying self- and non-self behaviors, it would also be useful to identify the type of malicious behavior is being classified. Knowing the type of attach can be useful for determining a mitigation strategy. Therefore, we propose to use *error correcting output codes* (ECOC) to strategically learn and combine binary classifiers to solve multi-class problems [32]. The general intuition behind ECOC is to break the multi-class problem into dichotomies, where classifiers are trained on different combinations of classes, but each problem is a binary classification task. A testing time the data is matched to a class by finding the closest code word (i.e., Hamming distance). As an example consider 7 classifiers trained on

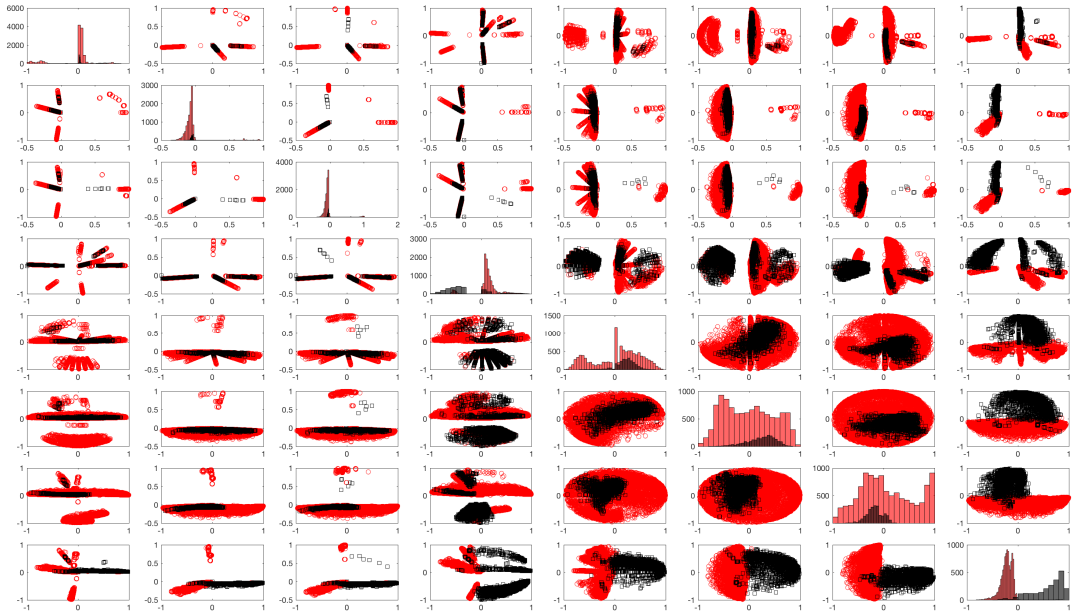
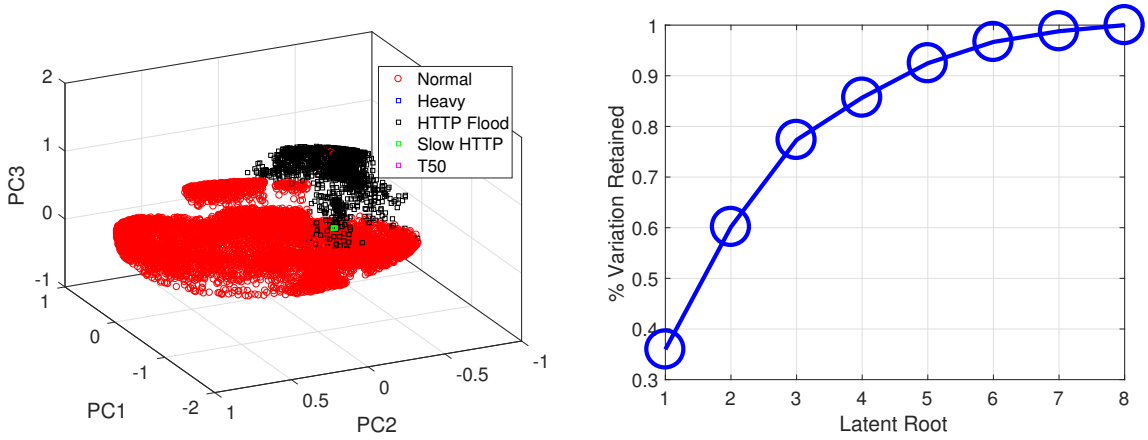


Fig. 3. Feature by feature scatter matrix plot where each feature is plotted w.r.t. all of the other features. The red and black points represent normal (self) and malicious (non-self), respectively.



(a) 3D projection of user data

(b) Cumulative % of variation in each eigenvector

Fig. 4. Visualization of the user/computer data in a 3D space by using principal components analysis (PCA). The multiple types of attacks that were run against a computer/user cluster together in the 3D space, which is a promising result if we are to build a classifier. Furthermore, the regardless of the attack type, the points still cluster together in the same region of feature space. Note there is a limited amount of data for some of the attack classes. The figure on the right shows the cumulative percentage of variation retained in each of eigenvectors.

different binary problems (based on 4 classes in the original data) that are defined in the matrix $\mathbf{C}$ [18]:

	h_1	h_2	h_3	h_4	h_5	h_6	h_7
ω_1	0	0	0	1	0	1	1
ω_2	0	0	1	0	0	0	0
ω_3	0	1	0	0	1	0	1
ω_4	1	0	0	0	1	1	0

where ω_k is a class and h_m is a classifier. Given $\mathbf{x}$, if the classifier's predict $\mathbf{h} = (0, 1, 1, 0, 1, 0, 1)$ then the respective Hamming distances are 5, 3, 1, and 5 to the code words in the table (i.e., the rows). Therefore, ECOC would choose ω_3 as the label for $\mathbf{x}$. The distances are calculated as follows: let $\mathbf{C} \in \{0, 1\}^{K \times M}$, K be the number of classes (i.e., types of attacks+1) and M be the number of classifiers then the class that ECOC selects is the one such that

$$\omega^* = \arg \min_{j \in [K]} \left\{ \sum_{i=1}^{M} |\mathbf{h}(i) - \mathbf{C}(j, i)| \right\}$$

D. Mitigation of Threats with Intervention

The threats were mitigated through the use of a software agent that runs in the background on protected computers. This software agent receives signals from the machine learning engine in the SPA and performs protective actions once the if a threat is detected in a computer, application or user. To reduce the false alarms in non-self malicious user behavior detection, we adopt Challenge-Response approach by requiring each user to provide the correct answers to a priori selected set of questions as shown in Figure 5. Then at runtime, if the SPA detects any non-self user behavior, it will ask the user to answer one or more random questions from the list as shown in Figure 5. If the user did not answer one of the question correctly during a specified period of time, an alert is generated and a signal is sent to the software agent on the affected machine to take the appropriate action. These actions include but are not limited to locking user account, disabling the user account, locking the machine, shutting down the computer, rebooting the computer, etc. If the user answers the questions correctly, the event will be logged and the user and/or the application model will be retrained on the new behavior. The questions and answers for each user are stored in a secure central database inside the SPA server.

IV. Experiments

This section presents the experimental design and results of the proposed SPA implemented using ECOC with SVMs on the data collected from our data SRFs.

A. Practices Followed

The classification results are presented in terms of a confusion matrix, and also report the accuracy, sensitivity and F-score for the binary classification problem. A confusion matrix is a table that is used to describe the performance of a classifier on a set of test data for, which the true values are known. The rows represent the class of the prediction that our model made and the columns are the true prediction. This allows us to observe the types of mistakes our model commits and the class which the model makes the mistake on. The tables are populated from 5-fold cross validation, where the k-fold splits are performed using stratified sampling. The accuracy is the number of fraction of correct classification to total number of items classified, the sensitivity is the true positive rate, and the F-score is the harmonic mean of the precision and recall [33].

B. Preliminary Results

1) Attacks on Users/Computers: For the attacks on users we first collected normal operational data for two users running on two Windows 10 computers that were also set up as web servers. We then collected data for those users as each of the following attacks was run on their respective computers using their accounts:

1) *HeavyLoad* – HeavyLoad was run on the computers in order to simulate an attack/malicious program that maximizes the usage of the computer's resources. The resources maximized by the program are CPU usage, GPU usage, memory usage and hard disk usage. Examples of malware that can cause this kind of behavior include malicious programs that force a computer to join a botnet that mines bitcoins, a malicious program which uses the affected machine as part of a distributed denial of service attack amongst others.
2) *HTTP Flood* – Using a program called LOIC (Low Orbit Ion Cannon) we flooded each of the computers with thousands of HTTP packets simulating a denial of service attack, which are common specially in important and/or high profile servers such as government servers and bank servers.
3) *Slow HTTP* – Using the SlowHTTPTest tool in Kali Linux, a version of Linux that contains multiple attack tools, we ran the Slowloris attack against the computers. The Slowloris attack is an application layer denial of service attack that opens as many connections to a web server as possible and keeps them open as long as possible. It does this by creating as many connections as it can and for each of these connections it only sends a partial request. The connections are kept open by sending subsequent HTTP headers through the connections but never completing the request. This has the effect of eventually filling the server's connection pool causing it to then reject legitimate connections. This is another variant of a denial of service attack which does not require a flood of packets. This attack is used to perform a denial of service attack while at the same time circumventing any flood detection technologies the victim might have in place.
4) *T50* – Using the T50 tool in Kali Linux, a version of Linux that contains multiple attack tools, we flooded the computers with network packets of various protocols including: TCP, UDP, ICMP, IGMPv2, IGMPv3, EGP, DCCP, RSVP, RIPv1, RIPv2, GRE, ESP, AH, EIGRP and OSPF. This attack is similar to the HTTP flood but the key difference is that it combines packets of multiple protocols so that even if a firewall is blocking packets for some protocols, the attack can still work.

TABLE I
Accuracy, sensitivity and F-score of the SPA trained on a self- vs non-self task.

	Computer #1	Computer #2
Accuracy	99.51	99.81
Sensitivity	98.99	98.46
F-score	99.16	99.01

Figure 7 shows the results of the SPA for defending users data on two different computers. The green and red squares represent correct and incorrect classifications, respectively. First, the accuracy for both the multi- and binary-class problem is 99%+; however, one of the attacks (T50) is not

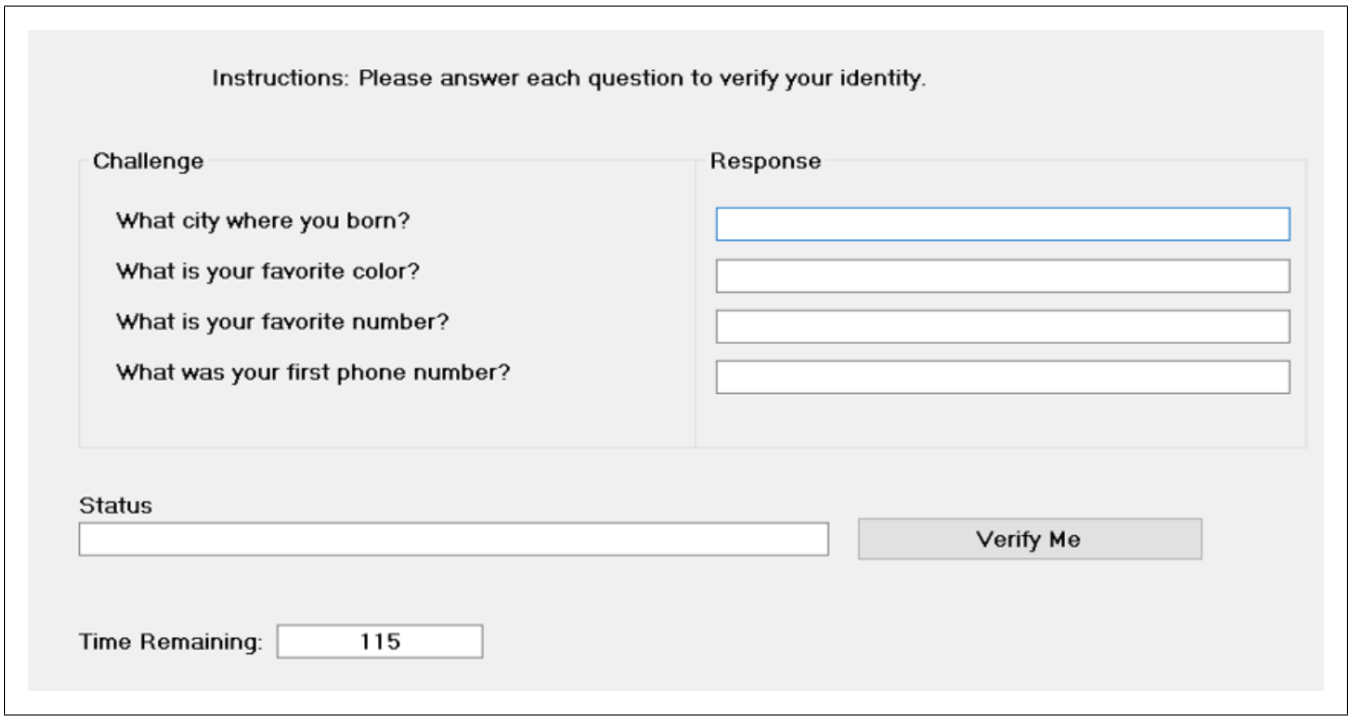


Fig. 5. Mitigation of threats GUI that prompts the user to answer several questions to verify that the computer/application is not compromised.

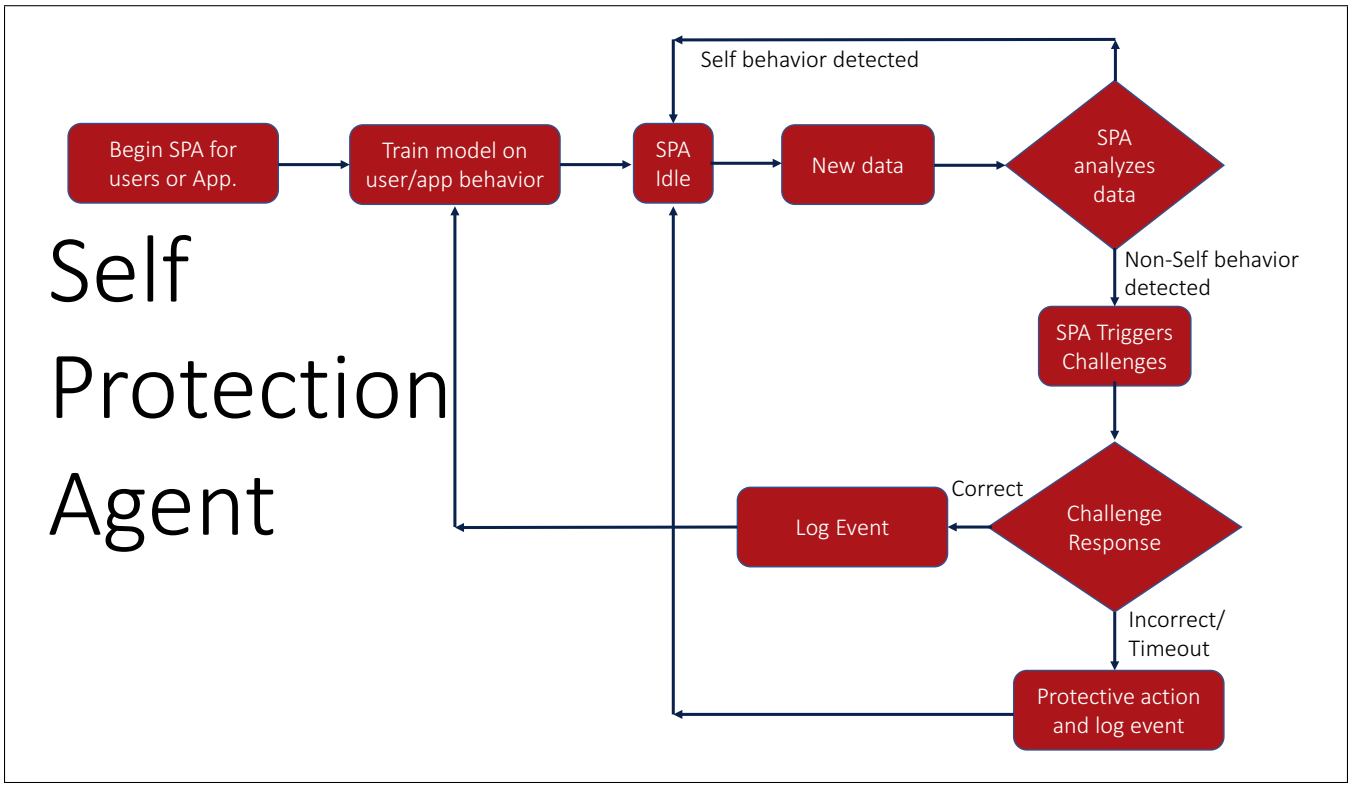
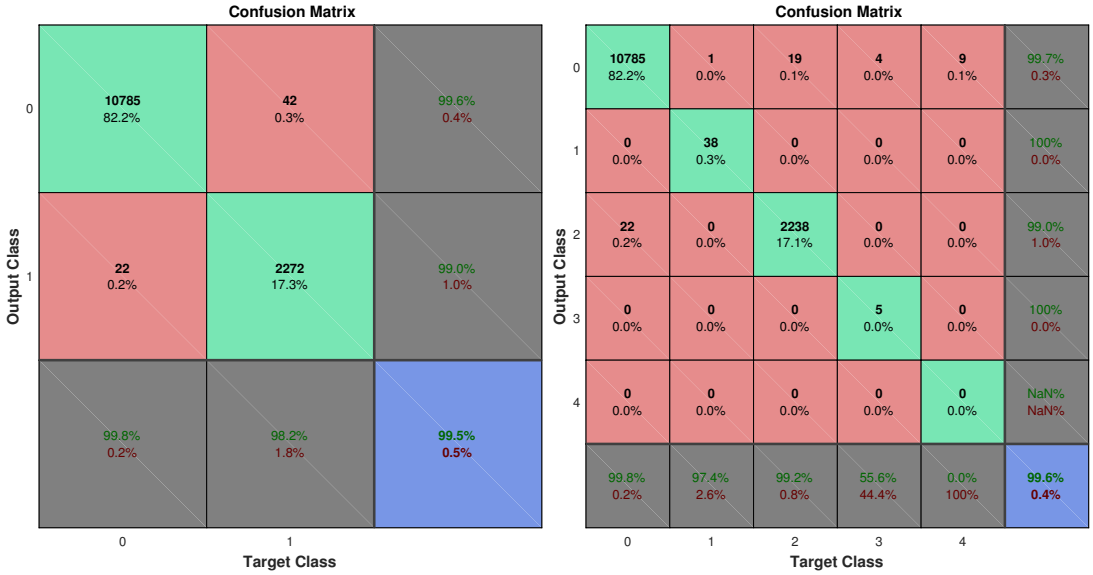


Fig. 6. High level block diagram describing the functionality of the Self-Protection Agent.



(a) Computer #1 with a binary classifier

(b) Computer #1 with ECOC

(c) Computer #2 with a binary classifier

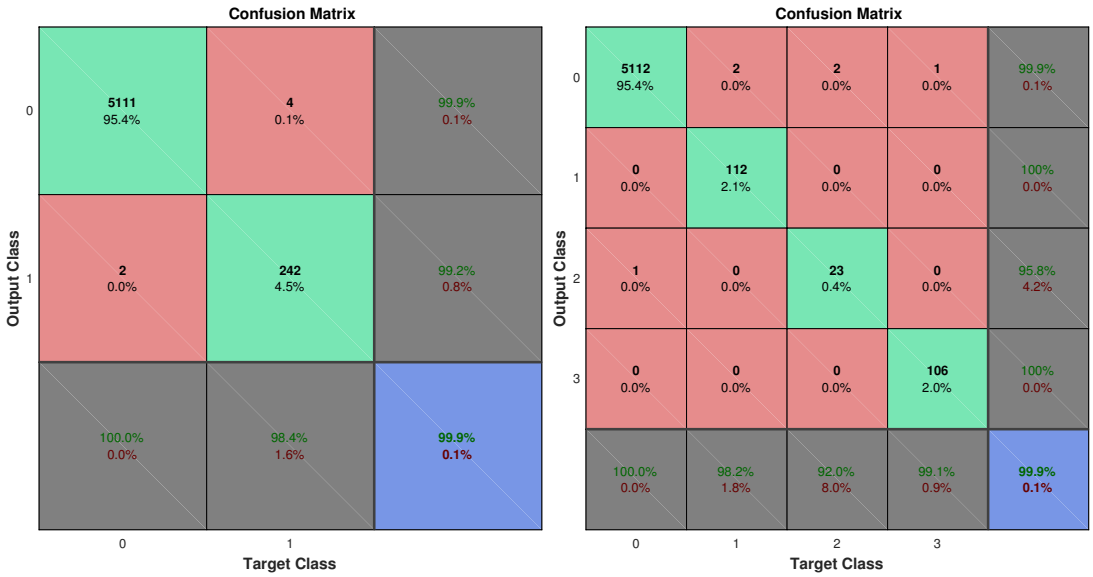
(d) Computer #2 with ECOC

Fig. 7. Visualization of errors being made by a binary and multi-class classifier on data collected from users/computers viewed as a confusion matrix. The classes are defined as follows: (0) normal, (1) heavy load, (2) HTTP flood, (3) slow HTTP, and (4) T50. Note that the data were collected from two computers; however, data for a slow HTTP attack were not available.

able to be learned likely due to the small sample size for the first computer (see Figure 7(b)). That is to say there are only 9 samples from the class, and the HTTP flood has nearly 2.2k samples. Such an under-representation can make it difficult for the classifier to learn the attack [34]–[36]. It is worth noting that computer #2 was able to detect this attack; however, computer #2 was not trained on the slow HTTP attack due to the availability of data. For both computers we observe very favorable results in terms of the correct identification of normal (self) and malicious (non-self). The binary classifiers' accuracy, sensitivity and F-score are shown in Table I.

2) Attacks on Applications: For the attacks on applications, we decided to test the SPA using an application that is commonly used by a large number of users, thus we decided use Mozilla Firefox, a commonly used open source web browser, for this test. We first collected normal operational data for the web browser. Next, we collected operational data while we had Firefox render pages with malicious JavaScript, in order to analyze it. The following types of malicious JavaScript scripts were run on the browser:

1) *Infinite JavaScript loop* – A JavaScript script that runs an infinite loop. This has the effect of making the process running the script hang.



(a) An Application SPA with a binary classifier

(b) An Application SPA with ECOC

Fig. 8. Visualization of errors being made by a binary and multi-class classifier on data collected from applications (Firefox) viewed as a confusion matrix. The classes are defined as follows: (0) normal, (1) heavy load, (2) HTTP flood, and (3) slow HTTP.

2) *JavaScript Fork Bomb* – A JavaScript script that runs a function that calls itself twice causing death by recursion.
3) *Heap of Death* – This script infinitely expands an array in memory greatly increasing the processes memory usage until it runs out of allocated memory.

One observation to immediately make is that these types of non-self attacks should be very easy to identify given the nature of the attack.

Figure 8 shows the results of the SPA for defending users data related to the Firefox application for a specific user. As expected the number of mistakes made by both the binary and multi-class SPA are very with detection rate typically greater than 99%, which is not surprising given the type of attack being run against the Firefox application. This result is observed for both the binary and multi-class SPA.

C. Mitigation of Threats with Intervention

The mitigation of threats was tested by running the attacks described in the previous sections against a protected computer. When the attacks were run against the computer and detected by the SPA, the appropriate actions were triggered. These actions include the initial challenge response prompt, which attempts to authenticate the user in order to avoid false positives. Once the user fails to authenticate successfully, the event gets logged for display in the dashboard shown in Figure 9 the software agent on the computer locks the machine so no further malicious actions can be performed by the attacker/intruder.

V. Conclusions

Defense against malicious cyber activities is one of the most important aspects facing a community where all devices are networked together, which allows for attackers to exploit vulnerabilities. This work presented an approach to classifying self- and non-self behaviors by developing a machine learning framework that was inspired by the human immune system. Our framework focuses on protecting users/computer and applications from several known types of attacks. A support vector machine is used within an ensemble of classifiers that are learned and combined using error correcting output codes (ECOC). This framework make it possible for incremental learning (if a incremental support vector machine is used) and allow for learning new types of attacks over time. We benchmarked out proposed framework on multiple types of non-self behaviors on computers and applications. The experimental results show that the proposed approach provides reliable classifications and the SPA is able to mitigate threats through Q&A once the threat is detected.

Our future work includes implementing the self-protection agent as a truly adaptive intelligent system by using machine learning tools from incremental learning [37]–[39].

Acknowledgements

This work was supported by the National Science Foundation under Award Numbers NSF CNS-1624668. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the funding agencies.

References

[1] M. Snider and E. Weise, "500 million yahoo accounts breached." http://www.usatoday.com/story/tech/2016/09/22/report-yahoo-mayconfirm-massive-data-breach/90824934.

[2] R. Hackett, "Quest diagnostics breach exposes health data of 34, 000 customers." http://fortune.com/2016/12/13/quest-diagnostics-data-breach-health/.

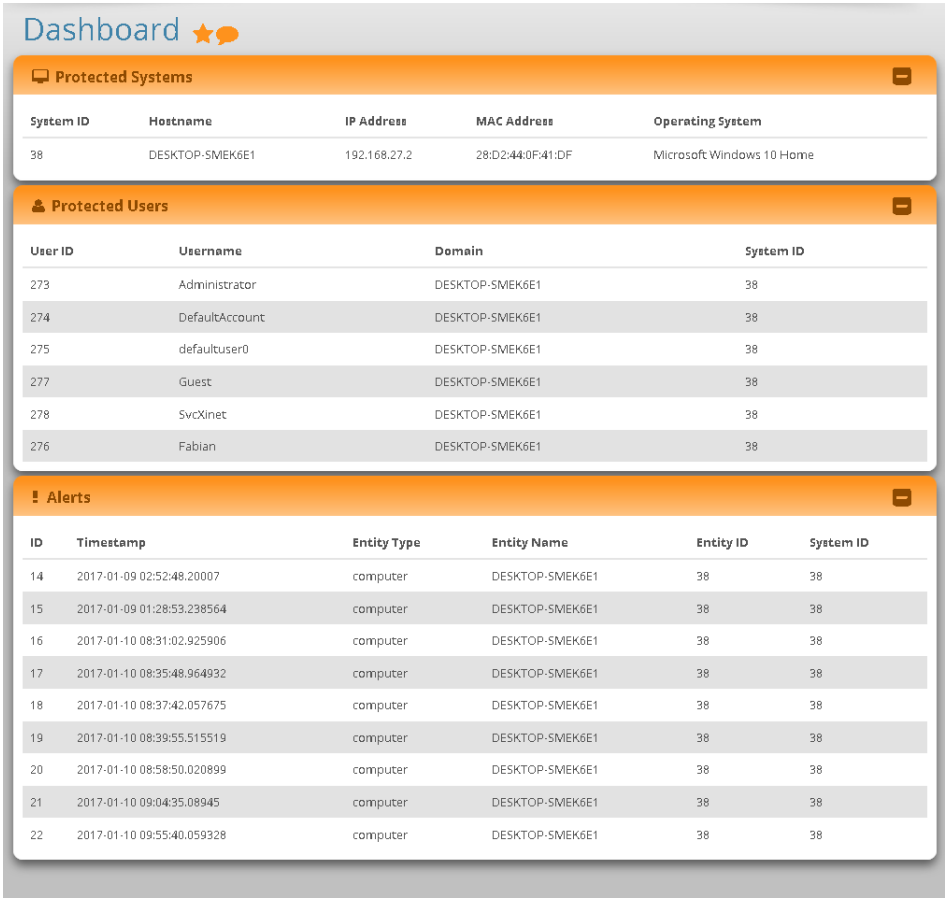


Fig. 9. SPA Dashboard

[3] E. Weise, "360 million myspace accounts breached." http://www.usatoday.com/story/tech/2016/05/31/360-million-myspaceaccounts-breached/85183200/.

[4] S. Hariri, B. Khargharia, H. Chen, J. Yang, Y. Zhang, M. Parashar, and H. Liu, "The autonomic computing paradigm," *Cluster Computing*, vol. 9, no. 1, pp. 5–17, 2006.

[5] M. Parashar and S. Hariri, "Autonomic computing: An overview," *Unconventional Programming Paradigms*, pp. 257–269, 2005.

[6] C. Tunc, F. Fargo, Y. Al-Nashif, S. Hariri, and J. Hughes, "Autonomic resilient cloud management (arcm) design and evaluation," in *International Conference Autonomic Resilient Cloud Management (ARCM) Design and Evaluation*, pp. 44–49, 2014.

[7] H. Alipour, Y. B. Al-Nashif, P. Satam, and S. Hariri, "Wireless anomaly detection based on IEEE 802.11 behavior analysis," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 10, pp. 2158–2170, 2015.

[8] 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, pp. 1153–1176, 2016.

[9] J. Sametinger, J. Rozenblit, R. Lysecky, and P. Ott, "Security challenges for medical devices," *Communications of the ACM*, vol. 58, no. 4, pp. 74–82, 2015.

[10] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier, 2005.

[11] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[12] J. J. Walker, T. Jones, and R. Blount, "Visualization, modeling and predictive analysis of cyber security attacks against cyber infrastructure-oriented systems," in *2011 IEEE International Conference on Technologies for Homeland Security (HST)*, pp. 81–85, Nov 2011.

[13] J. Ng, D. Joshi, and S. M. Banik, "Applying data mining techniques to intrusion detection," in *2015 12th International Conference on Information Technology - New Generations*, pp. 800–801, April 2015.

[14] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri, "Self-nonself discrimination in a computer," in *Proceedings of 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 202–212, May 1994.

[15] Y. Al-Nashif, *Multi-level Anomaly Based Autonomic Intrusion Detection System*. PhD thesis, Tucson, AZ, USA, 2008. AAI3336560.

[16] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[17] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. John Wiley & Sons, Inc., 2nd ed., 2001.

[18] L. I. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms*. John Wiley & Sons, Inc., 2004.

[19] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?," *Journal of Machine Learning Research*, vol. 15, pp. 3133–3181, 2014.

[20] Y. Freund and R. Shapire, "A short introduction to boosting," *Journal of Japanese Society for Artificial Intelligence*, vol. 14, no. 5, pp. 771–780, 1999.

[21] R. Polikar, "Ensemble based systems in decision making," *IEEE Circuits and Systems Magazine*, vol. 6, no. 3, pp. 21–45, 2006.

[22] Y. Ren, L. Zhang, and P. N. Suganthan, "Ensemble classification and regression-recent developments, applications and future directions," *IEEE Computational Intelligence Magazine*, vol. 11, no. 1, pp. 41–53, 2016.

[23] V. Vapnik, *The Nature of Statistical Learning Theory*. Springer-Verilag, 2nd ed., 1999.

[24] V. Vapnik, *Statistical Learning Theory*. Wily-Interscience, 1989.

[25] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine Learning*, vol. 46, pp. 389–422, 2002.

[26] S. Sonnenburg, G. Schweikert, P. Philips, J. Behr, and G. Rätsch,

"Accurate splice site prediction using support vector machines," *BMC Bioinformatics*, vol. 8, 2007.

[27] J. R. Cole, Q. Wang, E. Cardenas, J. Fish, B. Chai, R. J. Farris, A. S. Kulam-Syed-Mohideen, D. M. McGarrell, T. Marsh, G. M. Garrity, and J. M. Tiedje, "The ribosomal database project: improved alignments and new tools for rRNA analysis," *Nucleic Acids Research*, vol. 37, pp. 141–145, 2009.

[28] A. Ben-Hur, C. S. Ong, S. Sonnenburg, B. Schölkopf, and G. Rätsch, "Support vector machines and kernels for computational biology," *PLoS Computational Biology*, vol. 4, p. 10, 2008.

[29] B. Schlköpf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, 1st ed., 2001.

[30] J. Platt, *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*. MIT Press, 1998.

[31] P. Laskov, C. Gehl, S. Krüger, and K.-R. Müller, "Incremental support vector learning: Analysis, implementation and applications," *Journal of Machine Learning Research*, vol. 7, pp. 1909–1936, 2006.

[32] T. G. Dietterich and G. Bakiri, "Solving multiclass learning problems via error correcting output codes," *Journal of Artificial Intelligence Research*, pp. 263–286, 1995.

[33] J. Foster, S. Krone, and L. Forney, "Application of ecological network theory to the human microbiome," *Interdisciplinary Perspectives on Infectious Diseases*, 2008.

[34] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[35] H. He and E. Garcia, "Learning from imbalanced data," *IEEE Transactions on Data and Knowledge Discovery*, vol. 12, no. 9, pp. 1263–1284, 2009.

[36] N. V. Chawla, N. Japkowicz, and A. Kolcz, "Editorial: Special issue on learning from imbalanced data sets," *Sigkdd Explorations*, vol. 6, no. 1, pp. 1–6, 2004.

[37] R. Polikar, L. Udpa, S. S. Udpa, and V. Honavar, "Learn++: an incremental learning algorithm for supervised neural networks," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 31, no. 4, pp. 497–508, 2001.

[38] G. Ditzler, G. Rosen, and R. Polikar, "Incremental learning of new classes from unbalanced data," in *International Joint Conference on Neural Networks*, 2013.

[39] G. Ditzler, M. Muhlbaier, and R. Polikar, "Incremental learning of new classes in unbalanced datasets: Learn++.UDNC," in *Multiple Classifier Systems*, Lecture Notes in Computer Science, N. El Gayar et al., eds., vol. 5997, pp. 33–42, 2010.