AFRL-RI-RS-TR-2016-162

THUTMOSE – INVESTIGATION OF MACHINE LEARNING-BASED INTRUSION DETECTION SYSTEMS

BAE SYSTEMS INFORMATION AND SECURITY

JUNE 2016

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

AFRL-RI-RS-TR-2016-162 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
MISTY K. BLOWERS
Work Unit Manager

/ S /
ROBERT KAMINSKI for
WARREN H. DEBANY, JR.
Technical Advisor, Information Exploitation
and Operations Division
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* **OMB No. 0704-0188**

1. REPORT DATE *(DD-MM-YYYY)* JUNE 2016	**2. REPORT TYPE** FINAL TECHNICAL REPORT	**3. DATES COVERED** *(From - To)* SEP 2013 – NOV 2015

4. TITLE AND SUBTITLE THUTMOSE – INVESTIGATION OF MACHINE LEARNING-BASED INTRUSION DETECTION SYSTEMS	**5a. CONTRACT NUMBER** FA8750-13-C-0278
	5b. GRANT NUMBER N/A
	5c. PROGRAM ELEMENT NUMBER 63788F
6. AUTHOR(S) Anania, Mark; Corbin, George; Kovacs, Matthew; Nelson, Kevin; Tobias, Jeremy	**5d. PROJECT NUMBER** THUT
	5e. TASK NUMBER MO
	5f. WORK UNIT NUMBER SE
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) BAE Systems Information and Security 581 Phoenix Drive, Building 798 Rome, NY 13441	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RIGB 525 Brooks Road Rome NY 13441-4505	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFRL/RI
	11. SPONSOR/MONITOR'S REPORT NUMBER AFRL-RI-RS-TR-2016-162

12. DISTRIBUTION AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. PA# 88ABW-2016-2984
Date Cleared: 17 JUN 2016

13. SUPPLEMENTARY NOTES

14. ABSTRACT

In support of Air Force objectives to improve the Offensive and Defensive cyber-capabilities of the war fighter, this project endeavored to study learning systems researched and developed for cyber defense of network resources. Specifically, intrusion detection systems that were built with machine learning operations were studied to understand: the research behind the approach, the data they were designed to protect, the features processed, the algorithms used and the degree to which they were resistant and resilient to experimentally induced adversarial data drift. The results of this work provide deep insight into the strengths and weaknesses of the studied learning systems while operating within an adversarial environment. This insight will enable the design and development of future machine learning-based intrusion detection systems (ML-IDS) to be more hardened and effective in defending our nation’s networked resources. The experimentation results will aid in selecting or designing stronger algorithms, choosing better features, and more effectively monitoring resources. The toolset produced to run the experiments may be re-used and enhanced to make designing and testing of these future defenses faster and more effective.

15. SUBJECT TERMS

Machine Learning, Cyber Space, Intrusion Detection

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
					MISTY K. BLOWERS
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	92	**19b. TELEPHONE NUMBER (*Include area code)*** **N/A**

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 SUMMARY

In support of Air Force objectives to improve cyber capabilities of the war fighter, this project endeavored to study learning systems researched and developed for cyber defense of network resources. Specifically, intrusion detection systems (IDSs) that were built with machine learning (ML) operations were studied to understand: the research behind the approach, the data they were designed to protect, the features processed, the algorithms used and the degree to which they were resistant and resilient to experimentally induced adversarial data drift. The results of this work provide deep insight into the strengths and weaknesses of the studied learning systems while operating within an adversarial environment. This insight will enable the design and development of future machine learning-based intrusion detection systems (ML-IDS) to be more hardened and effective in defending our nation's networked resources. The experimentation results will aid in selecting or designing stronger algorithms, choosing better features, and more effectively monitoring resources. The toolset produced to run the experiments may be re-used and enhanced to make designing and testing of these future defenses faster and more effective.

In the course of this research, the team had the opportunity to co-author a book chapter and two papers as well as present the findings from this project at several technical conferences. The book chapter and two papers are included in the appendices.

2.0 INTRODUCTION

2.1 Purpose

The Thutmose project supported the Air Force Research Laboratory's (AFRL) mission to enhance cyber capabilities. Military computer networks are constantly under attack from adversaries who wish to compromise the integrity, confidentiality, or availability of system resources. In order to combat these attacks, intrusion detection systems are often put in place to monitor network and system resources for signs of malicious activity. Most commonly used IDSs are designed to detect signatures or patterns indicative of known attacks. However, these systems are rarely able to detect zero-day attacks or even slight modifications made to known attacks made by intelligent adversaries. For this reason, much research is being done to incorporate the field of machine learning into intrusion detection. Machine learning is a branch of artificial intelligence (AI) which focuses on the creation of models or rules which generalize and represent known data. An ML-IDS would either learn a model representing the normal behavior of a system and detect deviations from this model, or learn patterns from known attacks that could be generalized to new ones as well. These systems have shown considerable success in research environments, and it appears likely that they will begin to become much more common in operational systems. The focus of this effort was on the study of ML-IDSs and their inherit vulnerabilities and weaknesses in order to assess their potential defensive capabilities.

2.2 Background

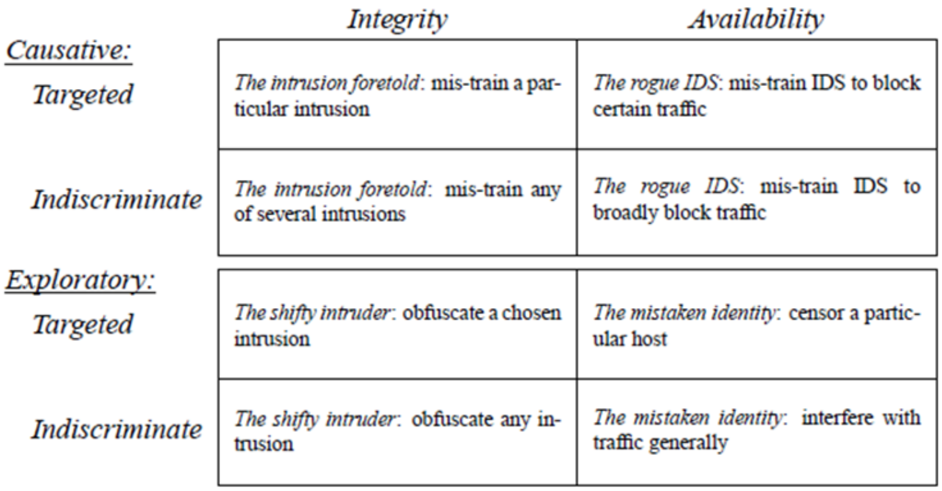
AFRL undertakes ongoing activity that requires development, implementation, integration, and evaluation of new techniques and software in the cyber domain. Therefore, there is a need to support capabilities that would support operational and mission requirements.

2.3 Scope

This effort was heavily research oriented, focusing on an investigation of previous academic research done in the field of ML-IDS, as well as actual implementations of IDSs which employ ML techniques. Key questions about these systems were established and were the focus of the initial investigation. The goal of the effort was to develop a methodology to measure the security of these ML algorithms when under attack in an adversarial environment, while identifying inherit strengths and weaknesses of these ML models.

Barreno et al. created a taxonomy to categorize potential attacks an adversary may employ against an ML system. Table 1 below is reproduced from the paper "The security of machine learning."

Table 1: Taxonomy of attacks against ML systems with examples



	Integrity	Availability
Causative:		
Targeted	*The intrusion foretold*: mis-train a particular intrusion	*The rogue IDS*: mis-train IDS to block certain traffic
Indiscriminate	*The intrusion foretold*: mis-train any of several intrusions	*The rogue IDS*: mis-train IDS to broadly block traffic
Exploratory:		
Targeted	*The shifty intruder*: obfuscate a chosen intrusion	*The mistaken identity*: censor a particular host
Indiscriminate	*The shifty intruder*: obfuscate any intrusion	*The mistaken identity*: interfere with traffic generally

This taxonomy was designed to address how an attack can affect the ML system. An attack has three dimensions: the Influence, the Security Violation and the Specificity of the attack. Influence is either causative or exploratory; the Security Violation attacks either integrity or availability; and the Specificity is either targeted or indiscriminate.

Integrity of the system is a measure that demonstrates the authenticity of data as having not been altered from origination or otherwise corrupted by malicious or accidental means. An Integrity attack is destructive in nature. Availability is a measure that represents how readily the data is accessed and used as intended by authorized users for intended and authorized purposes. An availability attack is a denial of service, either of specific resources or large-scale across a network.

The approach adversaries use may be causative, in which they take actions to bring about changes in the learning model through influence over the training data, or else exploratory in which their actions simply probe/investigate for potential weaknesses that can be exploited with another action. These causative and exploratory actions can be of two forms: targeted and indiscriminate. Targeted and indiscriminate attacks differentiate in their specificity and scope.

This effort primarily investigated targeted causative attacks against integrity, also known as adversarial drift, as this type of attack has been addressed very little in existing research. Under this type of attack, an adversary attempts to cause a specific point which is otherwise classified as anomalous by the learning model to be misclassified as normal through influence over the training data. Initial experiments focused on evaluating the extent to which an adversary could affect the algorithm's model and the amount of control an adversary would require over the training data in order to cause misclassifications in the resulting learning model.

The most popular forms of ML-IDS utilize anomaly-detection, a method in which algorithms first build a model of normalcy using benign network traffic and then evaluate new traffic against the model to detect behaviors or patterns deviating from this established normalcy. These methods have been shown to be effective at identifying potential threats to the system. Anomaly detection is often the popular choice over other machine learning techniques such as binary classification because it is often difficult to attain recent and representative malicious samples on which to train the models. For these reasons, this effort focused specifically on anomaly detection algorithms.

These activities will assist AFRL/RIGB with future operations.

3.0 TECHNICAL FACTORS OF EXPERIMENTATION

3.1 Methods, assumptions, and procedures

Based on the vast amount of research done to incorporate machine learning into intrusion detection systems combined with the inability of signature-based IDSs to detect unknown attacks, it is assumed that ML-based IDSs will soon become much more prevalent in operational settings. However, it is also known that ML tends to have several significant weaknesses, including a high false alarm rate and an ability to be manipulated by an adversary inserting strategic data. Therefore, studying the security and weaknesses inherit in ML-IDSs was considered an important research platform. Specifically, we wanted to study the vulnerability of ML systems to adversarial drift, a technique by which an adversary slowly inserts benign data that, over time and across multiple retrain iterations, causes the learning models to drift towards accepting previously anomalous points.

A series of questions about the systems studied was generated. The answers to these questions were deemed necessary as they would drive experimental direction and allow us to more accurately assess the potential defensive weaknesses in the IDSs. The questions included the following:

- What are the specifics of the machine learning algorithm implemented by the system? (If it uses a neural network, is it an Elman network, Kohonan network, etc.?)
- What type of data does it use? (Network-based or host-based? Packet headers or payloads? System calls?)
- How does it do feature selection or dimensionality reduction? Does it employ a kernel function?
- How much data is used to train the initial classification model?
- How is the classification model re-trained? Is the existing model updated or is it replaced with a new one?
- How often is the model re-trained and how much new training data is minimally required?

Our research approach began by identifying those machine learning algorithms which were most prominent in the published literature and in available open-source implementations of ML-IDS. After these algorithms were determined, the simplest ones that were also found to be common in the literature were selected for further, more in-depth study. During this stage, we attempted to find the most common answers to the above questions for each algorithm. The idea was to begin with the simplest algorithms and progress to those which were more complicated, applying lessons learned from the simpler cases along the way.

In an authentic network, an intruder would have limited access to information about an ML-IDS which has been deployed. While we certainly have a concern for external threats, the worst case scenario is an insider threat or an intruder who has already gained access to the internal network. We base our initial assumptions on this perspective. In order to have a worst case scenario baseline where a potential adversary would have near full control over the IDS and all associated and requisite resources, several assumptions were made to start with. Again the idea was to start with the simpler worst-case scenario, and to slowly remove assumptions in subsequent experiments, applying lessons learned from the simpler cases along the way.

We operated under what is known as the contamination assumption. This means that the adversary has the ability to insert points that will be used during retraining of the ML-IDS. Model drift is often accounted for through the use of retraining and online learning, so it is not unreasonable to assume an adversary could take advantage of this window to insert data. In our work, we also limited this assumption to create a slightly more realistic case in which insertion points must be classified as benign by the existing classifier in order to be included. We assume that the adversary would wish to remain undetected by including only points that are not flagged as suspicious and can be allowed into the re-training dataset via normal traffic through the monitoring ML-IDS. The contamination assumption is necessary if the adversary desires to induce drift in the models and force the ML-IDS to consider a point as normal that previous models would have otherwise identified as anomalous and blocked.

In initial experiments, we assume the adversary has full knowledge of and access to the IDS, its classification algorithms, the training data, and the results of classification. From this worst case scenario, we could potentially dial back the assumed knowledge and access privileges an adversary might have in order to gain a more realistic view of the measured weaknesses for each algorithm in an adversarial environment.

The main goal of our work was to develop a methodology using a framework to study and test ML algorithms which have or may be used in intrusion detection systems. This research was done in an effort to demonstrate the strengths and weaknesses of these algorithms in regards to their susceptibility to adversarial drift and offer suggestions for why and in what ways they can be effectively used in an IDS. We intended to create a generic, universal framework to allow for the simple incorporation of various additional algorithms using various training data sources. We wanted to be able to compare multiple algorithms, multiple implementations of similar or identical algorithms and to create repeatable experiments. Through experimental iterations, the framework should provide statistical measurements and analysis that better inform a security administrator.

We developed a Java application programming interface (API) based around Monte Carlo simulations to implement this framework and a prototype graphical user interface (GUI) to expedite experimentation with the API and enable automation of experimental runs. This API and GUI are detailed in section 3.2.4 **Monte Carlo Simulator**. The API allowed us to run a number of experiments using various ML algorithms and data sources which are discussed further in section 3.2.5 **Model Drift Experiments**.

The mathematical procedure defined in Figure 1 was developed to measure the number of points an adversary must insert in order to cause a misclassification on a particular point, and therefore the resistance of the ML algorithm to a targeted causative integrity attack or induced model drift. In the figure, *I* is equal to the list of injection points selected by the adversary, $g(x_0)_I$ is equal to the probability of x_0 according to the model with *I* injected into the training set, and γ is equal to the anomaly threshold.

```
initialize $I = \emptyset$
while $g(x_0)_I < \gamma$
        $x^* = \underset{x}{argmax}\, g(x_0)_{I+x} - g(x_0)_I \;\; s.t. \;\; g(x)_I > \gamma$
        add $x^*$ to $I$
return $|I|$
```

Figure 1: Experimental approach to determine effort required by an adversary to force misclassifications

A key part of this effort was to determine the most optimal method for calculating x^*, which is the point that causes the greatest model drift in the direction of the adversary's target point. A generic approach was desired that would allow a number of different machine learning algorithms to be quickly analyzed using our developed methodology. A number of different optimization strategies were researched and developed and are detailed in section 3.2.

3.2 Results and Discussion

3.2.1 Algorithm Selection. Several machine learning algorithms that were most commonly used in the research were identified. These included K-nearest neighbor (KNN), support vector machines (SVM), artificial neural networks (ANN), decision trees (DT), self-organizing maps (SOM), hidden Markov models (HMM), and Bayesian classifiers. The most common ML algorithms proposed for IDSs according to two separate literature reviews are summarized in Figure 2 and Figure 3.

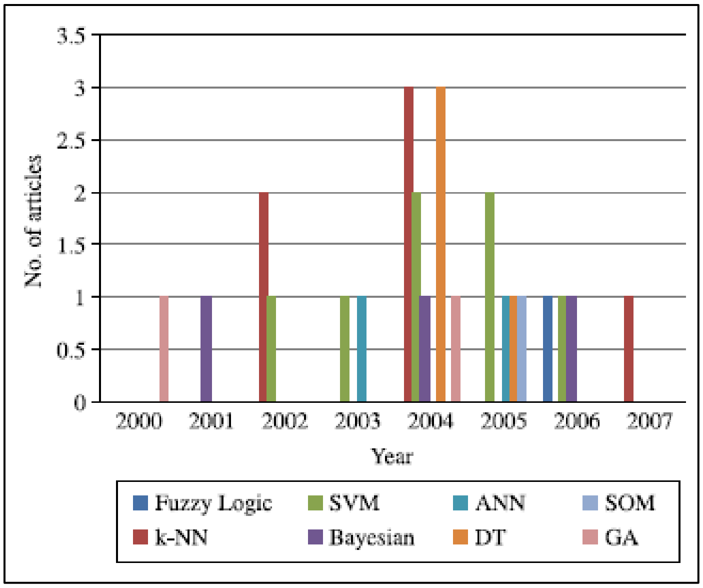
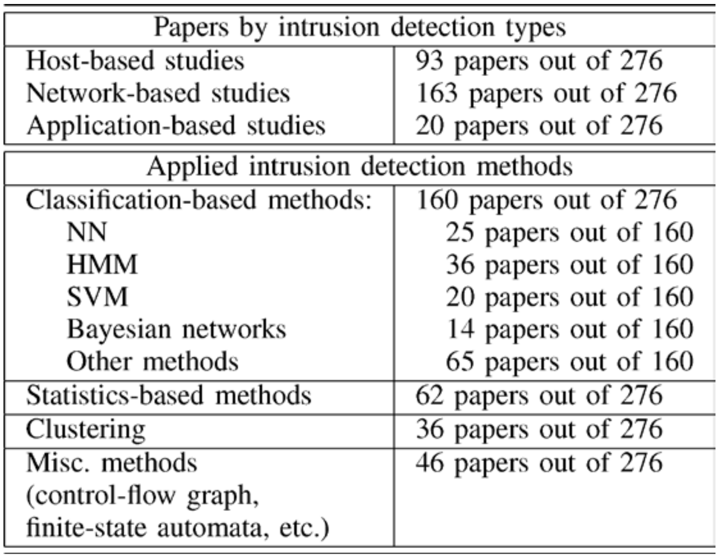


Figure 2: Year-wise distribution of articles for single classifiers according to (Tsai, Hsu, Lin, & Lin, 2009)



Papers by intrusion detection types	
Host-based studies	93 papers out of 276
Network-based studies	163 papers out of 276
Application-based studies	20 papers out of 276
Applied intrusion detection methods	
Classification-based methods:	160 papers out of 276
NN	25 papers out of 160
HMM	36 papers out of 160
SVM	20 papers out of 160
Bayesian networks	14 papers out of 160
Other methods	65 papers out of 160
Statistics-based methods	62 papers out of 276
Clustering	36 papers out of 276
Misc. methods (control-flow graph, finite-state automata, etc.)	46 papers out of 276

Figure 3: Details of surveyed papers according to (Tavallaee, Stakhanova, & Ghorbani, 2010)

Based on these findings, additional research, and findings from available real-world ML-IDS, we selected four algorithms, Centroid Anomaly Detection, HMM, SVM, and K-Means Anomaly Detection, to further explore initially. According to our research, these algorithms were among the most commonly explored as well as being three of the simpler ML algorithms to conceptualize, visualize, and understand. Upon selecting these algorithms, we began identifying and studying research papers which described IDSs using each algorithm to determine how they answered the questions posed above. Unfortunately, we did not have time to explore the additional identified algorithms within the scope of this effort.

The research papers studied primarily relied on two different sources of data. The first was network connection records and the second was system call traces. Much of the research utilized the dataset from the Third International Knowledge Discovery and Data Mining (KDD) Tools Competition as their source of network connection data. This data, referred to as the KDD Cup '99 dataset, consists of individual connections, each labeled as either normal or as a specific type of attack. It was created by processing the tcpdump portions of the 1998 Defense Advanced Research Projects Agency (DARPA) IDS Evaluation dataset. While it is widely used in the research, the KDD Cup '99 dataset has several known problems and is outdated. The system call traces used in much of the research come from the Basic Security Module (BSM) audit data portion of either the 1998 or 1999 DARPA IDS Evaluation data.

A variety of dimensionality reduction and data pre-processing techniques were discussed in the research. Some of the most common included normalization, principal component analysis (PCA), converting symbolic features to numeric, and converting system call traces to frequencies of individual calls.

3.2.1.1 Centroid Anomaly Detector. The first algorithm studied was the simple centroid anomaly detector. This algorithm is trained by simply finding the empirical mean of the training examples. An unlabeled data point is then tested by calculating its Euclidean distance to the mean, or centroid. If this distance is greater than a determined threshold, then the point is considered to be anomalous. Calculation of a data point's anomaly score is summarized below.

$$f(x) = \left\| x - \frac{1}{n}\sum_{i=1}^{n} x_i \right\|$$

Figure 4 shows an example of a centroid anomaly detector in two-dimensional space. The green points within the anomaly threshold would be classified as normal while the red points that fall outside of the threshold would be considered anomalous. Despite its incredible simplicity, this algorithm is popular in various security applications. We chose to run initial experiments with the centroid anomaly detector due to its low computation time, applicability, and its ability to be easily comprehended and visualized. Its primary purpose is to be used as a baseline experiment to demonstrate our approach.

We implemented this algorithm in Java and incorporated it into our API (discussed in section 3.2.4.1 ***Java API***) for experimentation.

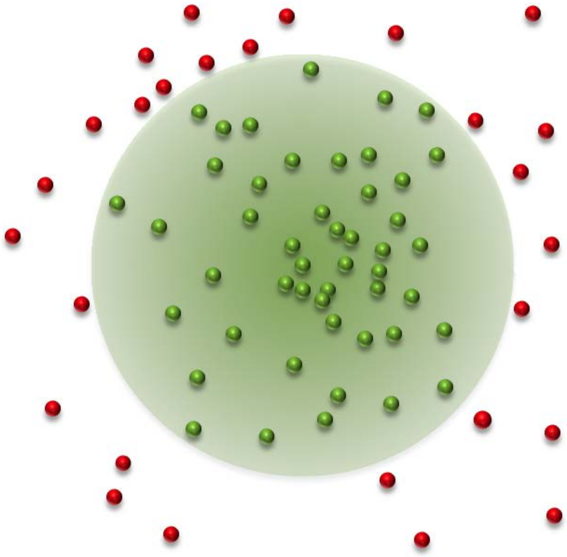


Figure 4: Centroid Anomaly Detector

3.2.1.2 Hidden Markov Model. The next algorithm examined was the Hidden Markov Model. An HMM is a machine learning model consisting of hidden states which follow the Markov property and have associated initial and transition probabilities. In addition, an HMM consists of observed variables, with each variable having a certain probability of occurring in each hidden state. Sequences of observed variables are used to train the HMM and learn the hidden state and observed variable probability matrices. Once learned, these probabilities are used to determine the probability score of new test observation sequences. Figure 5 displays a visualization of an HMM. Each arrow connecting an observed variable to a hidden state would have an associated probability, as would each arrow connecting the hidden states. This algorithm was chosen again due to its relative simplicity as well as its common appearance in ML-IDS research.

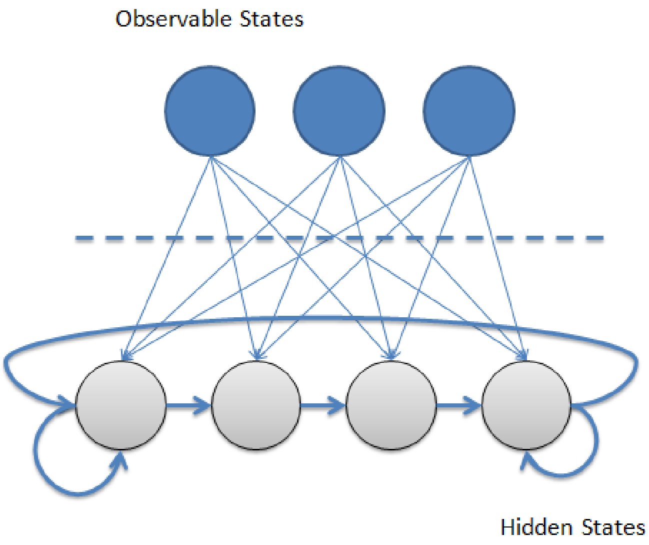


Figure 5: Hidden Markov Model

We utilized the open-source Java library JAHMM implementation of HMMs to incorporate this algorithm into our experimentation framework (discussed in section 3.2.4 **Monte Carlo Simulator**) for initial experiments. Additionally, we identified two available ML-IDS which implement and rely primarily on HMMs, SuStorID and HMMPayl, which are discussed in sections 3.2.2.1 SuStorID and 3.2.2.2 HMMPayl.

3.2.1.3 Support Vector Machine. The third algorithm investigated was the support vector machine. An SVM is generally a binary linear classifier which classifies an object based on which side of a dividing hyperplane it falls. The hyperplane is constructed to create maximum separation between the two classes in the training set. In cases where the two classes are not linearly separable, a kernel function such as the radial basis function (RBF), may be applied to map the data into a higher-dimensional space. Figure 6 shows a simple example of an SVM in three dimensional space.

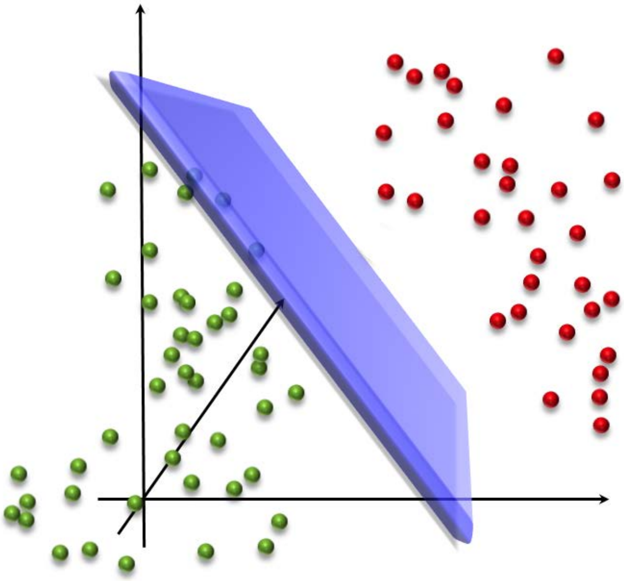


Figure 6: Support Vector Machine

Several variations of SVMs were described in the literature. The most common was the standard linear binary SVM which separated "normal" data from "attack" data. Also, slight modifications to this algorithm were proposed, such as the Robust SVM which introduces new slack terms to address the over-fitting problem. In addition, an anomaly detection method, or one-class SVM, was described which creates a hyperplane that separates the entire training set from the origin. If an unknown point is separated from the origin then it is considered normal, otherwise it is considered anomalous or an attack. Using SVMs to classify an unknown point as one of multiple classes was also suggested. A series of binary SVMs are created, and then organized into a decision tree. For example, an SVM may be created to separate classes A and B from C and D, another one to separate A from B, and another to separate C from D.

As we had previously decided to focus primarily on anomaly detection algorithms, we chose to investigate one-class SVMs. The LibSVM library implementation of SVMs was used to incorporate this algorithm into the experimentation framework (discussed in section 3.2.4 **Monte Carlo Simulator**). We also identified and experimented with an open-source ML-IDS called McPAD, which utilizes one-class SVMs and is discussed in section 3.2.2.3 ***McPAD***.

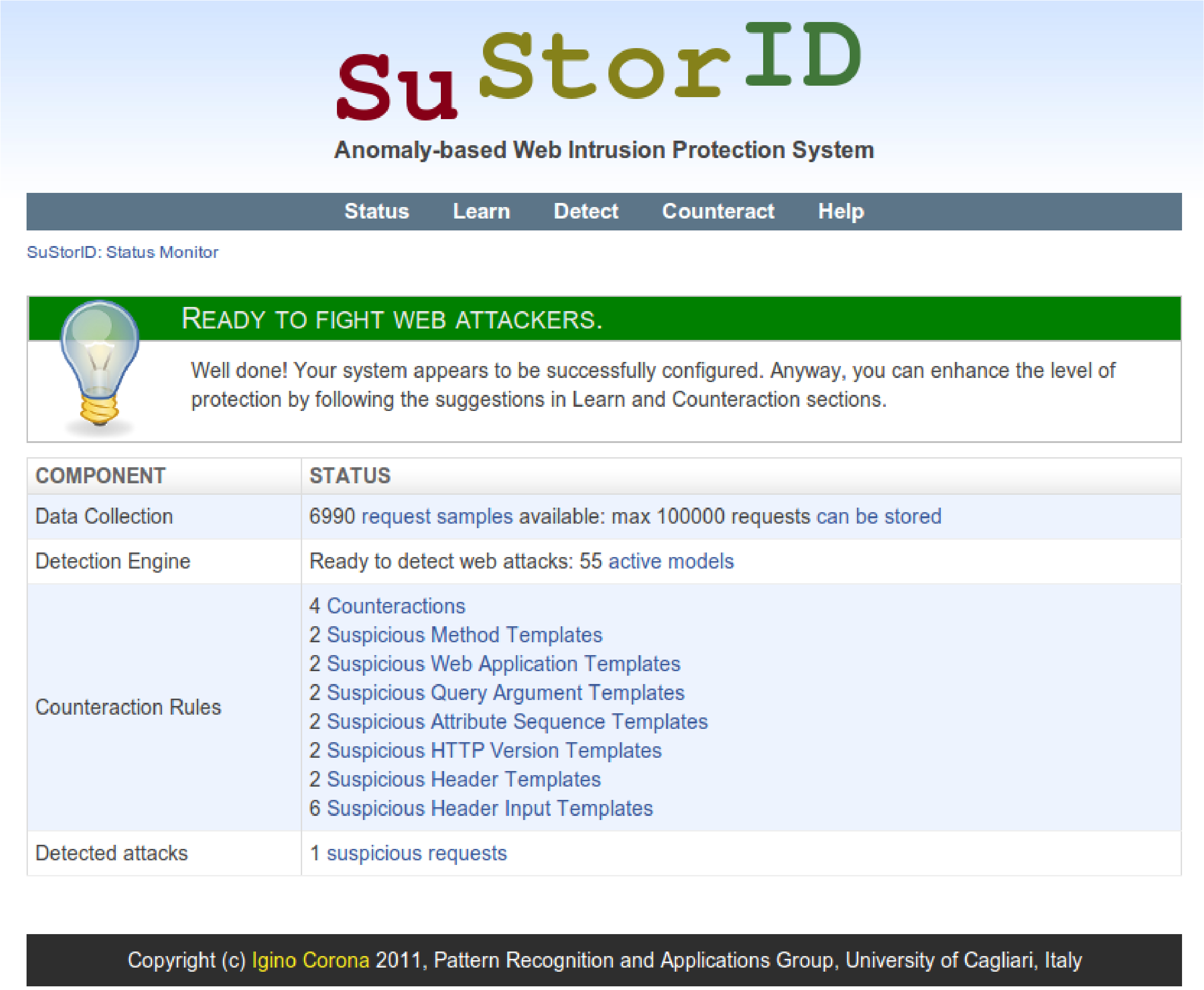
3.2.1.4 K-Means Anomaly Detector. As the research progressed, we additionally chose to investigate the K-Means anomaly detection algorithm. This algorithm operates by first clustering the training data using the K-Means clustering algorithm in the feature space. K-Means clustering groups the training data into a set number of clusters, K, by first placing K centroids in the feature space. Then, iteratively until a termination criterion is met, all points are assigned to the nearest centroid, and the centroids are re-centered to the mean of the assigned points. After the cluster centroids are determined using K-Means, a threshold bound is set around each such that a set percentage of the training points lie within the threshold. At test time, a test point is assigned to the nearest cluster centroid as is determined to either fall inside or outside of the threshold bound. The point is classified as anomalous if it is not inside the threshold and normal if it is within the threshold.

This algorithm is similar to the centroid anomaly detector, but the clustering step allows for multiple centers of density within the training data. This algorithm was chosen for its common occurrence in anomaly detection applications and its relative simplicity while still adding a layer of complexity to the centroid anomaly detector. Also, we desired an additional distance-based algorithm that operated in the feature space with which to test and verify our optimization methods (section

*3.2.1.5 Optimization Selection***.** The K-Means Anomaly Detection algorithm was implemented in Java and incorporated into our experimentation framework (discussed in section 3.2.4 **Monte Carlo Simulator**). The JavaML library was used to implement the K-Means clustering step.

3.2.2 Real World ML-IDS. A variety of real world implementations of IDSs that relied on ML were identified. Three of these systems were successfully incorporated into our experimentation framework (section 3.2.4 **Monte Carlo Simulator**) and used for model drift evaluation.

3.2.2.1 SuStorID. The first such system was SuStorID, an open-source host-based IDS for web services which uses Hidden Markov Models to perform anomaly detection. SuStorID is written in Python, utilizing the Django framework, and is coupled with the Apache-based ModSecurity web application firewall to gather training data and provide real-time counteractions. SuStorID's models are trained using recent HTTP requests and can be retrained at any time using the tool's interface. We installed and configured SuStorID on an Ubuntu VM and began initial analysis of the tool and its algorithms.



SuStorID: Status Monitor

READY TO FIGHT WEB ATTACKERS.

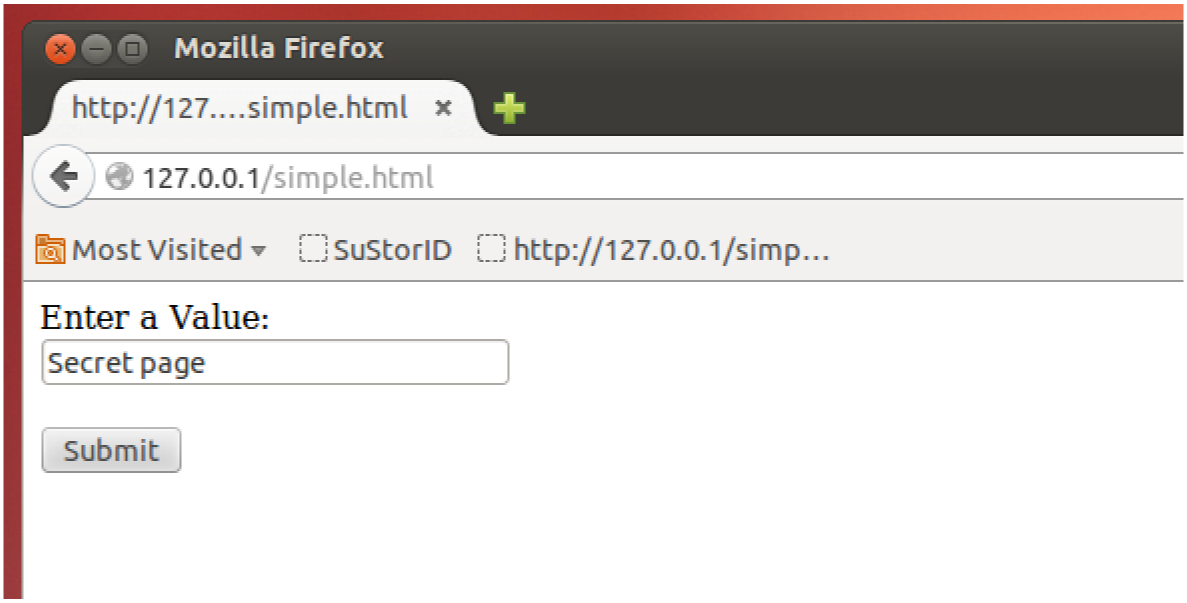
Well done! Your system appears to be successfully configured. Anyway, you can enhance the level of protection by following the suggestions in Learn and Counteraction sections.

COMPONENT	STATUS
Data Collection	6990 request samples available: max 100000 requests can be stored
Detection Engine	Ready to detect web attacks: 55 active models
Counteraction Rules	4 Counteractions 2 Suspicious Method Templates 2 Suspicious Web Application Templates 2 Suspicious Query Argument Templates 2 Suspicious Attribute Sequence Templates 2 Suspicious HTTP Version Templates 2 Suspicious Header Templates 6 Suspicious Header Input Templates
Detected attacks	1 suspicious requests

Copyright (c) Igino Corona 2011, Pattern Recognition and Applications Group, University of Cagliari, Italy

Figure 7: SuStorID's user interface

3.2.2.2 Technologies Used. In order to test SuStorID, a web application was required to be placed on the apache web server to be protected by the IDS. Initially a simple web page containing an HTML form (Figure 8) was created to meet minimum requirements for SuStorID training processing.



Enter a Value:

Secret page

Figure 8: Simple HTML Test Page

We next incorporated SuStorIDS into our experimental framework (section **3.2.4 Monte Carlo Simulator) to a**llow for the simple processing of end-to-end experiments. To support the training process, automated generation of training data needed to be produced as well as an automated means of creating and inserting drift and test points into the training datasets. Several approaches were tried. Table 2 below provides a concise summary of each tool and how it was tried within the experimental framework.

Table 2: Methods utilized during experimentation with SuStorID to generate training and test data

JMeter	Java based functional and performance testing tool which uses XML to specify the interfaces to automate testing. It can be used to test Web (HTTP and HTTPS), SOAP/REST, FTP, and database via JDBC, LDAP, Message-oriented middleware (MOM) via JMS, Mail (SMTP, POP3 and IMAP), MongoDB (NoSQL), Native commands or shell scripts, TCP. Was too slow for large volume of training dataset traffic generation, but it is used to initiate the training sequence for SuStorID due to the high level of control provided with the simple XML format it uses.
Java	Used the HTTP libraries to create and send HTTPRequests. This is the method finally used to generate the traffic to be used in the training datasets due to the ease with which new data can be generated quickly while maintaining prefect control over the contents of the packets. This is key in isolating the one pertinent feature examined in the experiment.
Edit DB tables	Java use of JDBC libraries to add rows to DB storing traffic in order to add requests.

	This was tried and replaced due to the performance impacts of coordinating several steps such as: locking and unlocking the tables; shutting down and bringing up the IDS; monitoring the IDS to make sure the locking and shutdown/restart steps occurred after all requisite traffic had been received.
Edit/replay PCaps	Using a modeled HTTPRequest inside a PCap (network packet capture) file to be a template for generated traffic for training data, insertion data and test data. This method was tried and held for use later in the examination of other IDS's. For the purposes of SuStorID it was very slow performance due to the PCap editing software and the PCap replay software currently being so slow to execute with faster options available.
Mergecap	Wireshark comes with a host of tools for manipulating PCaps files. Mergecap is one used to merge multiple packets from 2 or more PCap files into one larger PCap file for ease of processing. Some IDSs required the training data to be in one large PCap file for one or more steps.
Mechanize-perl	Perl scripts using HTTP library to generate HTTPRequests. This was used after a more comprehensive website based upon WordPress was developed as a more realistic web application for testing. When SuStorID's unreliability became an issue for the more realistic website, this approach was abandoned as not needed any longer.
Selenium	Scripting to click through on the browser to generate traffic for training. Very slow execution as it ran in a browser and would not have been usable for later tests proposed to be run on the HPC cluster due to its reliance on a GUI in the form of the web browser. Scripting to click through on the browser to generate traffic or training.
Apache Bench (ab)	Would not install properly, nothing accomplished with it. This is a risk with Open Source Software.
Httperf	Great large volume of traffic generated, not enough control of what precisely was IN the traffic to be usable for selective creation of insertion and test points. We required control over every feature examined by the IDS and some of them were not accessible using Httperf
Tsung	Too complicated to configure and make work for multiple types of traffic.
Web Scarab	Too slow and resource intensive resulting in averaging one HTTPRequest generated per second.
HTTPReq Generator	Missing functions for execution to work. Did not provide the function to specify some features monitored by IDS.
Metasploit (from Kali Linux)	An easy way to generate web server HTTP traffic is to use Metasploit to run the WMAP module. WMAP probes the server port which can rapidly generate thousands of packets using the available interface. These packets are essentially scripted so are only useful for generating the bulk of the normalized traffic, but not the specific insertion and test traffic.

3.2.2.3 HMMPayl. The next ML-IDS investigated was HMMPayl. HMMPayl is an open-source network IDS, written in Java, and developed by PRA Lab. It utilizes HMMs trained using the Baum-Welch algorithm to detect anomalous packets. HMMPayl inspects network packet payloads represented as byte strings, using a sliding window to extract n-grams from these byte strings, and uses these n-grams as the feature set to train its models. In an effort to increase classification accuracy, HMMPayl uses an ensemble of HMMs, combining the results from each to make its predictions. Figure 9 outlines the architecture of HMMPayl.

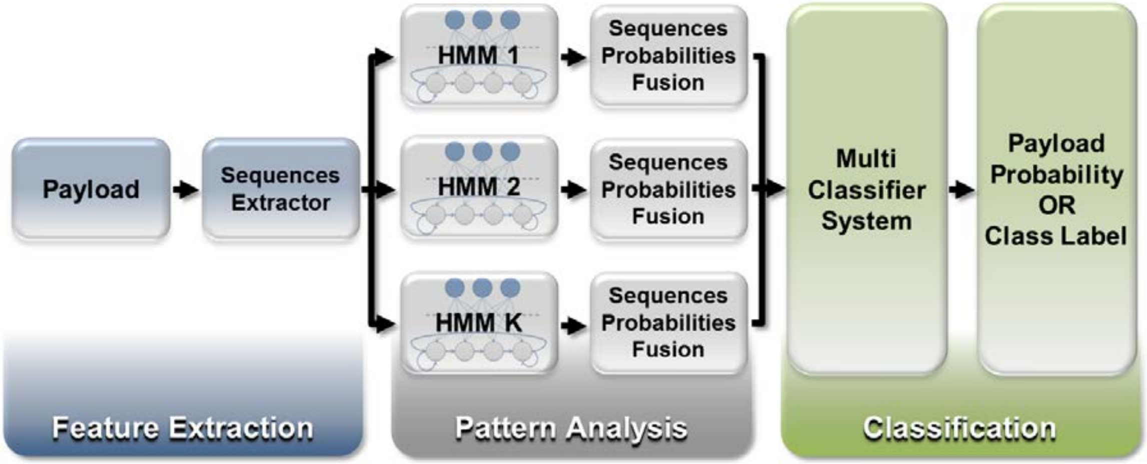

Payload
Sequences Extractor
HMM 1
HMM 2
HMM K
Sequences Probabilities Fusion
Sequences Probabilities Fusion
Sequences Probabilities Fusion
Multi Classifier System
Payload Probability OR Class Label
Feature Extraction
Pattern Analysis
Classification

Figure 9: HMMPayl process overview

In order to facilitate experimentation with HMMPayl, we first incorporated it into our framework (section 3.2.4 **Monte Carlo Simulator**). Since HMMPayl is written entirely in Java, as is our framework, this task was simplified. The key parts of the ML-IDS are the ML model training step, and the test step. The code responsible for these tasks was located in HMMPayl's source and extracted to our own class. This code relies on the existence of a PCap file for training and then either monitors traffic for testing, or uses a separate testing PCap file. In order to support this training and test process, a method for the automatic generation of packets needed to be produced. We reused the code developed for experimentation with SuStorID to send custom HTTP requests that were then collected into PCaps and used for training/testing. HMMPayl uses the jpcap library to capture packets, create PCaps, and to parse them to create its training and test sets. A bug existed in the jpcap library utilized by HMMPayl that was causing errors during processing. The issue was fixed by modifying jpcap's native C code and re-making the files. Additionally, HMMPayl's Java code was modified because it did not properly close its captors after it was done with them, which was leading to errors during iterative experiments. This allowed for initial end-to-end experiments to run successfully.

The method of creating the training sets by sending custom HTTP requests and capturing the packets proved to require too much processing time in order to effectively run iterative experiments. Even if working directly with existing PCaps, processing the files and extracting the packet payloads was computationally expensive. However, examining the code revealed that all HMMPayl uses to train its models are the byte strings that make up the packets payloads, which are just sequences of numbers. We therefore stripped out the code from HMMPayl which processes the PCaps and extracts the packet payloads so that it could work directly with lists of numbers. This allows us to extract the payloads from our training files prior to the experimental runs, and crafting insertion points becomes as simple as creating lists of numbers. This makes training more dynamic, as we can more easily choose which points from the training set to utilize and can store the training sets in other formats such as CSV or in a database.

Additionally, HMMPayl's original code required the trained HMM models to be saved to a file after training, and then reloaded each time a new point was tested. This repetitive file IO led to increased computation time and slowed down iterative experiments. Therefore, we modified HMMPayl's source code to store the trained HMM models in memory, saving significant time during experimental runs.

3.2.2.4 McPAD. The next ML-IDS analyzed was McPAD (Multiple classifier system for accurate Payload-based Anomaly Detection), another open-source network IDS written in Java and developed by PRA Lab. McPAD is a breed of ML-IDS which analyzes the payloads of web traffic it is monitoring. This analyzed payload is within the application layer of the OSI model. The analysis tries to establish whether or not the payload is malicious assuming that the byte distribution of a malicious payload differs from that of a normal payload. McPAD uses a 2 nu-gram analysis of the payload byte distribution when building the models to use for discriminating new incoming normal payloads from potentially malicious ones. This analysis is an approximation of the n-gram analysis commonly used in text-classification. A 2 nu-gram consists of two bytes from the packet payload byte string that are nu bytes apart. A sliding window across the payload is used to find all of the 2-nu grams. These are then clustered to decrease the feature set size from 2562 dimensions down to a specified number of dimensions (160 by default). Using the paradigm of Multiple Classifier Systems, McPAD leverages an ensemble of Support Vector Machines to perform the processing of both model development and intrusion detection using the 2 nu-gram analysis of byte distribution in payloads of incoming traffic to the webserver or hosted web applications it is protecting. McPAD trains multiple SVM models utilizing different values of nu for each when building its 2 nu-grams and combines the results of each at test time. Figure 10 presents a simplified overview of McPAD.

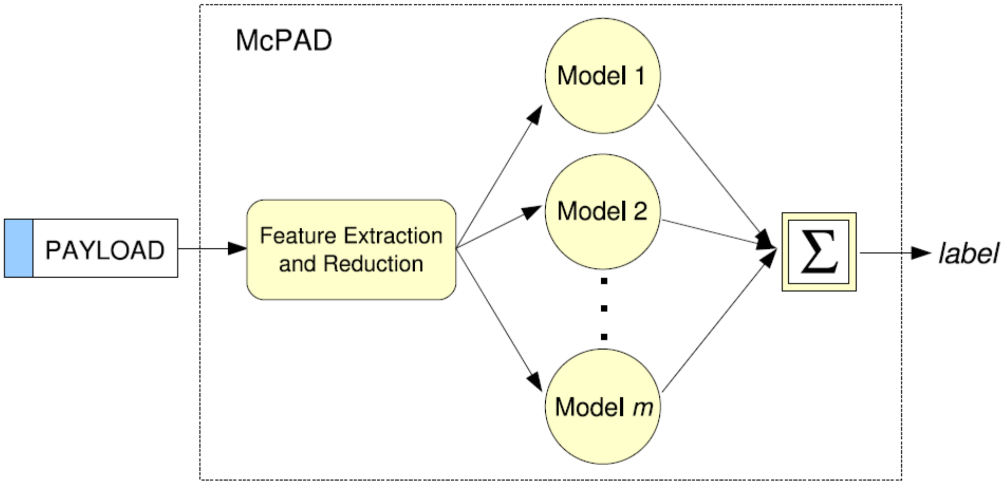


Figure 10: McPAD process overview

The first step towards analyzing McPAD's susceptibility to model drift was to incorporate it into our experimental framework (section 3.2.4 **Monte Carlo Simulator**). Like HMMPayl, McPAD is also written entirely in Java, so this task was again simplified. The code responsible for training and testing the SVM models was located in McPAD's source and extracted to our own class. This code extracts data from a PCap file to train its models and then either tests monitored live traffic against the models, or uses a separate test PCap file. In order to support this training and test process, we needed to produce a method for the automatic generation of packets. We reused the code developed for experimentation with SuStorID to send custom HTTP requests that were then collected into PCaps and used for training/testing. McPAD also relies on the use of the jpcap library, so again the bug in jpcap's native C code had to be fixed. Several other modifications to McPAD's source code were also necessary in order to properly process end-to-end experiments. The original McPAD code would shut down any time a packet was encountered that it could not cast to an IP packet, which occurred often. The code responsible for this was located and modified to continue processing. Additionally the code was edited so that during live monitoring, McPAD would actually use the network interface controller (NIC) specified rather than always using the second one in the device list. Also, like HMMPayl, McPAD did not properly close its captors after it was done with them, which was leading to errors during iterative experiments. This was fixed. Code was also added so that during testing, the probability score of the packet with the lowest probability score was tracked. This is because a single "test point" may span multiple packets, and if a single packet fell below the anomaly threshold, then essentially the test point was detected. These modifications allowed us to run initial experiments.

The method of creating the training sets by sending custom HTTP requests and capturing the packets again proved to be too computationally expensive to effectively run iterative experiments. Similar to HMMPayl, however, McPAD only requires the byte string from the packet payloads to train its models. We therefore stripped out the code from McPAD which processes the PCaps and extracts the packet payloads so that it could work directly with lists of numbers representing the byte strings. This allows us to extract the payloads from our training files prior to the experimental runs, craft insertion points by simply creating lists of numbers, dynamically create training sets, and store training sets in a variety of formats.

Additionally, McPAD's original code required that the training data (after 2 nu-gram analysis), the feature cluster information, and the SVM models were all saved to a file during training. The cluster information and the SVM models then had to be reloaded at test time. During iterative experiments, this repetitive file IO added unnecessary computation time. Therefore, we modified McPAD's source code so that this information was all stored in memory during experimental iterations, increasing the speed of experiments.

3.2.2.5 PESCAN. Another ML-IDS investigated was PESCAN, a malware analyzer developed by BAE Systems. PESCAN uses a byte scanner to detect executable code embedded in network traffic and inspects its structural properties for patterns indicative of malicious executables. PESCAN is incorporated into the Suricata framework, a popular open-source network IDS developed by the Open Information Security Foundation (OISF). To perform the learning system training phase, PESCAN analyzes portable executable meta-data for structural features, numerical properties, and import lists. It trains decision trees, a popular ML algorithm, using the feature criteria mentioned above, and utilizes them for the classification of traffic as malicious or not. PESCAN examines portable executables found in the traffic and processes them to compute a PEScore (risk metric) in order to aid in identifying portable executables that are likely to be malicious. We installed and configured PEScan on an Ubuntu VM. However, due to limited access to useable data and the significant difference in the type of data this ML-IDS monitors from the others studied, we did not find time to further experiment with it during this period of performance.

3.2.2.6 Additional. As ML-IDS is still a new and emerging technology, few actual implementations, particularly open-source implementations, are currently available for experimentation. During our research, however, we identified several applications of ML for cyber security purposes which were either open-sourced or in use by corporations. These technologies are summarized in Table 3. We did not further explore these applications under this effort because their implementations/code were not made available to us or because they were deemed less appropriate / lower priority than those discussed in the previous sections and time did not allow it. They do, however, provide compelling evidence that machine learning is becoming more prevalent in cyber security and further justify our research.

Table 3: Machine learning cyber security applications identified but not further researched

Application	Description
Clonewise	Utilizes Random Forest classification to detect package clones and infer security vulnerabilities from those clones that are out of date. Used by Debian Linux.
Zozzle	Tool developed by Microsoft Research to perform static analysis of JavaScript code on a site and determine whether or not it is malicious. Uses Bayesian classification.
MLIDS	The Machine Learning Intrusion Detection System (MLIDS) is a tool funded by the DoD to detect attacks against High Level Architecture (HLA) and Distributed Interactive Simulation (DIS) simulation environments. Uses Support Vector Machines.
Nova	Creates a set of honeypots, and then uses ML to identify patterns of hostile network traffic. Uses K-Nearest Neighbors for classification.
ORCA	Oak Ridge Cyber Analytics (ORCA) is a suite of tools that include ML capabilities for analyzing network traffic to detect zero-days and other attacks and for distinguishing the real attacks highlighted by IDS alerts from expected events. Funded by Lockheed Martin and was included in their experimental Defense and Self-Healing Network.
HMM-Web	Developed by PRA Lab to detect server side attacks against web applications. Uses an ensemble of Hidden Markov Models to detect anomalies.
TotalADS	The Total Anomaly Detection System (TotalADS) is a framework for detecting host-based anomalies. Analyzes execution traces and log files using Sequence Matching (SQM), Kernel State Modeling (KSM), and Hidden Markov Models.
Cynomix	Tool developed by Invincea and funded by DARPA to analyze malware and detect malicious programs. Uses ML to compare the similarity of unknown programs to known malware strains.
CylancePROTECT	Endpoint security tool developed by Cylance that uses ML to detect and prevent the execution of advanced malware and persistent threats. Collaborating with Dell to be integrated into the Dell Data Protection | Endpoint Security Suite Enterprise.
Exabeam	Platform developed by Exabeam performs user behavior analysis using unsupervised ML. Uses anomaly detection to detect cyber-attacks and insider threats.

3.2.3 Optimization Selection. In order to properly test our approach for measuring the susceptibility of ML-IDSs and their associated algorithms to model drift, we had to first develop a method to find the value of x which maximizes $g(xo)I+x - g(x0)I$ s.t. $g(x)I > \gamma$ (Figure 1). That is, we wanted to develop an optimization algorithm to automatically generate the point which when added to the training set, causes the greatest amount of drift in the resulting ML model towards the desired test point. We initially developed optimization algorithms and heuristics that were based on detailed knowledge of the ML algorithms under study. However, these optimization approaches, while reasonably effective, required a thorough study of the ML algorithms and were generally only applicable to a single ML algorithm. This required an understanding and a time investment that a researcher may not have in a realistic environment. Therefore, to make our methodology more generic and applicable to future ML algorithms, we began to study universal, heuristic-based optimization approaches that were ML algorithm independent. The optimization approaches we studied and developed, both ML algorithm dependent and independent, are detailed in the following sections.

3.2.3.1 Centroid Anomaly Detector Optimization. The approach for generating the point which optimally drifts a centroid anomaly detector toward a target point is relatively straightforward and is, in fact, mathematically provable. The adversary determines the vector between the target anomalous point and the current centroid and inserts a point along this vector at a distance from the centroid equal to the anomaly threshold value. This approach is illustrated in Figure 11.

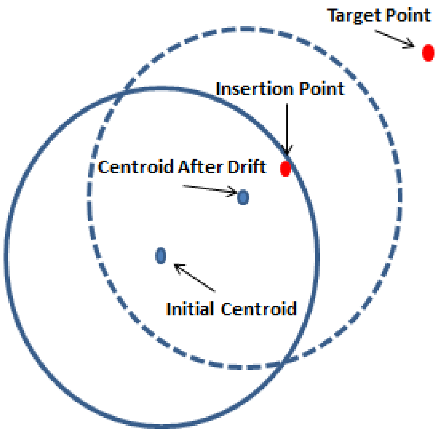


Figure 11: Illustration of optimal drift strategy against a centroid anomaly detector

This insertion point will appear normal, due to falling within the anomaly threshold, but upon retraining of the ML model will drift the centroid in the direction of the target point.

3.2.3.2 K-Means Anomaly Detector Optimization. A similar approach was developed for generating the point which optimally drifts a K-Means anomaly detector towards accepting a target point as normal. The adversary determines the cluster centroid that is nearest to the target point, determines the vector between the centroid and the target point, and inserts a point along this vector that is a distance away from the centroid equal to the anomaly threshold value. This causes an effect similar to that induced on the centroid anomaly detector. Upon retraining, the cluster is drifted towards the target point, until the point is eventually no long classified as anomalous. Additionally, the insertion points will appear normal due to falling within the threshold bound of the cluster.

3.2.3.3 HMM Optimization. Through an investigation of HMMs, we developed a fast method to determine the near-optimal points for inducing model drift. Since the sequence probabilities returned by an HMM are largely driven by the frequency of individual symbols in the training set, we chose to focus our approach on these frequencies. We wanted to target the specific n-grams from the test point that contained symbols common in the point but that also had low probabilities, meaning that they likely contained symbols uncommon in the training set. Therefore, to create the adversary's insertion points, for a set number of iterations we iteratively added symbols to the insertion point according to the equation below where T(x) is the number of times symbol x occurs in the test point and Tr(x) is the number of times x appears in the training set.

$$x^* = \underset{x}{\arg\max} \frac{T(x)}{Tr(x)^2}$$

Then, if the constructed insertion point is considered anomalous by the existing model, symbols in the point are iteratively replaced by the symbol occurring most frequently in the training set until the point is no longer flagged.

During our experiments with HMMs, various other optimization approaches were tested but were determined to be less effective for various reasons. These approaches are described in Table 4.

Table 4: Optimization strategies examined for generating optimal drift points for HMMs

Method	Explanation	Issues
Genetic algorithm. Fitness function is defined as the increase in test score of the test point when adding insertion point to the training set and retraining. Tried various methods including different mutation/cross-over methods and a variation that allowed for variable length solutions.	Popular and easy to implement machine learning optimization method. It is generic and requires little prior knowledge of the ML algorithms.	Requires retraining the model for each fitness calculation. This takes far too long for large data sets. Was not producing great results in a reasonable amount of time.
Particle Swarm Optimization. Fitness function is defined as the increase in test score of the test point when adding the point to the training set and retraining.	Popular and easy to implement machine learning optimization method. It is generic and requires little prior knowledge of the ML algorithms.	Requires retraining the model for each fitness calculation. This takes far too long for large data sets. HMMs consider individual character frequencies so distance between points is irrelevant, making this approach not highly effective for this purpose.
Test all permutations (with replacement) of the test point and find the one that has the lowest test score while also being above the anomaly threshold. If there isn't one above the threshold, replace one of the values with a random value from the training set.	Does not require retraining the ML models. HMMs consider character frequencies so designed to increase frequency of characters in test point.	For large test points, the time to test all permutations is much too high.
Incrementally add the character from the test point that occurs the least in the training set until the point is the length of the average point from the training set. If the point is detected, iteratively replace a character in the point with the most frequently occurring character in the train set.	Assumes full knowledge of the training data. Approach is based on character frequencies while not requiring testing all permutations of test point.	Often the character from the test point which occurs the least frequently in the training set also occurs infrequently in the test point. This means that it will occur in a small number of n-grams and have a small effect on overall probability of the point.
Incrementally add the n-gram with the lowest score from the test point. Score in calculated for an n-gram by summing the difference in number of occurrences in the training set and the test set for each character in the n-gram. If the point is detected, iteratively replace a character in the point with the most frequently occurring point in the train set.	HMMs consider sequences of characters, so an approach which adds sequences with low probabilities to the test point could improve results. The selected scoring method seemed to correlate with n-grams that had low probabilities yet contained common characters in the test point.	Experimental results were actually considerably worse than using selected method.

3.2.3.4 SVM Optimization. During our experimentation with SVMs, we developed a method to quickly determine the insertion point which will have the near-optimal drift effect. The SVM implementation that we chose utilized a Gaussian kernel, so the score of a test point is largely driven by its distance from the support vectors in the input space. Additionally, not all training points are considered support vectors, so we must attempt to ensure that the insertion point is chosen as a support vector in order to have an effect on the test score. With these factors in mind, we developed an initial method to generate insertion points that relies an in-depth knowledge of the SVM. For our chosen approach, we find the support vector nearest to the target anomalous point and insert the point along the vector between the two that is closest to the target anomaly without being flagged as anomalous. This method proved to be effective during initial experiments with training data sampled from a Gaussian distribution. However, it did not appear to extrapolate well to alternate data distributions.

During our experiments with SVMs, various other optimization approaches were tested but were determined to be less effective for various reasons. These approaches are described in Table 5.

Table 5: Optimization strategies examined for generating optimal drift points for SVMs

Method	Explanation	Issues
Particle Swarm Optimization. Fitness function is defined as the increase in test score of the test point when adding the point to the training set and retraining.	Popular and easy to implement machine learning optimization method. It is generic and requires little prior knowledge of the ML algorithms.	Requires retraining the model for each fitness calculation. This takes far too long for large data sets.
In the kernel space, find the point that is closest to the test point which also lies on the separation plane defined by the alpha values and the threshold. Then use particle swarm optimization to find the point in the input space which when kernelized lies closest to this point and is classified as normal.	Does not require retraining the ML models. The goal was that this point would push the separation plane in the direction of the test point the furthest.	When adding the point to the training set, it is not selected as a support vector during training, so has no real effect on the model.

3.2.3.5 Universal Optimization. While the algorithm-specific optimal point generation algorithms that we developed were relatively effective, we wanted to develop an approach that was more universal and able to quickly test multiple ML algorithms without requiring a deep understanding of their details. This would make our methodology for measuring the susceptibility of ML algorithms to model drift much more applicable to future algorithms we wish to analyze. For this reason, we investigated heuristic based optimization methods. Heuristic methods make few assumptions about the problem being optimized and can generally search large spaces of candidate solutions. These methods require a custom fitness function, or a measure of how good or applicable a potential solution may be. In order to make sure our implementations remained algorithm-independent, we wanted to develop a fitness function that relies only on knowledge of the resulting test scores of the target point and thresholds of the algorithms. Another benefit of these heuristic methods is that they do not require this fitness function to be differentiable.

3.2.3.5.1 Fitness Functions. We developed several different fitness measurements during our experimentation. The first that was relatively successful was to use the resulting test score of our target point after retraining the ML models with the potential point inserted into the training set. The optimization algorithm would then find the point that when added to the training set, maximized the score of the target point. While testing this method, however, we found that it was often the case that while the resulting test score of the target point would increase, the anomaly threshold would often increase at a nearly equal rate. This meant that the classification of the target point would never change, and it would remain anomalous to the ML models. To remedy this, we designed a fitness function that would use the difference between the resulting score of the target point and the resulting threshold. The algorithm would then find the point which minimized this gap to ensure that while the test score was increasing, the target point also appeared less anomalous to the ML models. If a point was found that caused the test score to become larger than the anomaly threshold, then it was selected automatically. If the test score of the potential insertion point made it appear anomalous to the existing classifier, then it was given a score of 0 and no longer considered. This method had improved results during experimentation. Additionally, it had the interesting side-effect that often the points being generated would focus primarily on lowering the anomaly threshold rather than increasing the test score of the target point.

3.2.3.5.2 Particle Swarm Optimization. The first heuristic based optimization approach that was used to automatically generate insertion points to induce model drift was Particle Swarm Optimization (PSO). PSO is an evolutionary algorithm that works by placing a certain number of particles (candidate solutions) into the feature space which then, for a certain number of iterations, travel to new locations searching for an optimal solution. Each particle's movement is dictated by mathematical calculations based on the particle's position and velocity, the best point seen by the particle, and the global best point seen by any particle. We chose this algorithm due to its wide-spread use and popularity and its proven success in a variety of optimization problem spaces. We utilized JSwarm, an open source Java library, and its implementation of PSO during our experimentation. The fitness function described above was implemented and used to measure the fitness of candidate points during the experiments. This optimization approach allowed for some success in initial experiments. However, it was noted that this approach would not be the most effective for algorithms like HMMs where the distance between points in the feature space is not highly correlated to test scores. The non-convexity in the feature space meant that this approach may not optimally converge to a solution as designed.

3.2.3.5.3 Genetic Algorithms. We next investigated genetic algorithms (GA) as a heuristic approach to generate points that optimally induce model drift. GAs are another evolutionary approach that attempt to mimic the process of natural selection. GAs begin with an initial population of potential solutions, stochastically selects the most fit members, combines these members in a process known as crossover, and introduces random mutations. This process of selection, crossover, and mutation is repeated iteratively for a set number of iterations or until an acceptable solution has been found. We chose GAs because they are a very widely used heuristic method and have proven success in a variety of optimization domains. Additionally, they appear to be less specialized than PSO, meaning that while GAs may be less suitable for some applications, they are more effective for a wider variety of problems. Therefore, they allow us a more universal approach, applicable to a variety of ML algorithms. During our experimentation, we implemented our own version of a genetic algorithm in Java. We chose to use fitness proportionate selection (roulette wheel selection), two point crossover, and random mutations. We also chose to ensure that the fit member from any population is guaranteed to make it to the next generation unchanged. We experimented with a number of different mutation rates and crossover rates and different mutation methods such as decreasing the mutation rate as the number of iterations increases and decreasing the mutation range as the number of iterations increases. Additionally, we developed and experimented with custom mutation and crossover techniques that allowed our population to be made up of members of varying lengths.

One of the major drawbacks of GAs is the computational time required to repeatedly evaluate the fitness functions. Particularly in our case, our chosen fitness function requires retraining ML models, which is an intensive process. In an attempt to mitigate this, we investigated Adaptive Fuzzy Fitness Granulation (AFFG). AFFG maintains a list of fuzzy granules, or groups of similar points whose fitnesses have already been computed. If an individual is sufficiently similar to one of these granules according to a fuzzy similarity analysis, then it is simply assigned the fitness of the granule. Otherwise, its fitness is calculated and a new granule is created. This cuts down on the number of explicit fitness calculations that must be processed. During our experimentation, we implemented AFFG in Java and incorporated it into our GA code.

3.2.3.5.4 Nelder-Mead Method. We additionally investigated the Nelder-Mead method as a heuristic approach for generating points that optimally induce model drift. In n dimensions, the Nelder-Mead method creates a simplex arranged of n+1 candidate solution points. It then iteratively replaces the point that has the lowest fitness with a point reflected through the centroid of the simplex. It also either expands or contracts along this line to either stretch of shrink the search space depending on the fitness of this reflected point. This has the effect of replacing "lower" points with "higher" points, moving the simplex "uphill" towards the best solution and contracting at the top rather than sliding back down. This algorithm was chosen because it is another popular and widely used technique. Also, it tends to require fewer fitness calculations than a GA, greatly decreasing the amount of required computation. During our experimentation, we implemented the Nelder-Mead method in Java. We also ran experiments utilizing the Apache Commons implementation of the algorithm. It was noted, however, that the Nelder-Mead algorithm tended to have similar weaknesses to PSO. In a non-convex feature space, the algorithm may not converge to an optimal solution.

3.2.3.5.5 Simulated Annealing. As the other heuristic optimization methods that we explored did not produce as optimal of solutions as we believed were possible, we next investigated Simulated Annealing (SA). SA choses an initial state (candidate solution), then iteratively selects a neighboring state and probabilistically decides to remain in the current state or move to the new state. This acceptance probability is based on the fitness score of the two states and the current temperature. The temperature decreases over the iterations, causing the algorithm to be less likely to select states with lower fitnesses in later iterations. In the early iterations, however, SA may select a state with a lower fitness in order to avoid local maximums. During our experimentation, we first used JAnnealer, an open source Java implementation of SA, but it did not offer all of the functionality that we required. Instead, we next used the AIMA3e library, an open-source Java library containing various optimization algorithms. In order to improve its implementation of SA, we modified it so that during optimization iterations, the best solution encountered was remember throughout, and returned at the end.

3.2.4 Monte Carlo Simulator. As mentioned above, we designed and developed a Java API and prototype GUI that were used to implement our framework and methodology and facilitate experimental runs. We chose to design our framework around Monte Carlo simulations, which is an experimental method that relies on repeated random sampling. This helps eliminate data specific results by finding averages across multiple runs, decreasing the variance in our results. To this end, an API was created, described in the next section, with interfaces that allow for simply running Monte Carlo experiments and adapting new algorithms and data sources for testing.

The framework allows for experiments to be run to identify precisely what effort an adversary would need to expend in order to force a misclassification on the selected test point through model drift created by the introduction of crafted insertion data into the training set. These experiments may be repeated for varying training set sizes, different percentages of control the adversary has over the retrain data, and varying algorithm-specific parameter values. For each configuration, multiple iterations are run with randomly sampled data from the selected data source and the results are aggregated and placed into graphs for further analysis. This provides the user with an overall picture of the resistance of the algorithm to adversarial drift and may be used to compare algorithms.

3.2.4.1 Java API. The API was designed to be flexible, allowing us to test various functionalities using Monte Carlo simulations with a variety of machine learning algorithms and data sources. The API was written primarily in Java due to its object oriented design, its abundance of existing libraries, and the team's skill set.

An abstract class, Algorithm, was created which contained the majority of the functionality required to implement a machine learning algorithm and begin testing with it. To implement an algorithm, the user simply must extend this class and implement the train, test, and classify methods. The train method receives a training set as a parameter in the form of a List of Lists of Numbers, which it will then use to create its learning models. The test method receives a single test point in the form of a List of Numbers, which it will then test against the learned models, and return a numerical test score. The classify method receives a single test point which it will compare against the learned models, and return either a 0 or a 1 indicating whether or not the point is anomalous. During our experiments, classes were created that extended this Algorithm class to implement each of the algorithms identified in section 3.2.1 Algorithm Selection as well as the IDSs SuStorID, McPAD, and HMMPayl.

An abstract class, DataStorage, was created to handle the functionality of storing data and making it available to the algorithms for training and testing. To implement a DataStorage, a user must extend the abstract class and implement the getRandomPoints and getAllPoints methods. The getRandomPoints method returns a given number of points randomly selected from the data source in the form of a List of Lists of Numbers. Its purpose is to allow for repeated Monte Carlo simulations which call for randomly selected training data sets. The getAllPoints method returns all of the points from the training set in the form of a List of Lists of Numbers and is designed for instances where randomness is not desired. During our experimentation, we implemented a number of different DataStorages. Table 6 details the different data sources that we were able to pull from using our DataStorage implementations.

Table 6: Data sources implemented and incorporated into Java API

Data Source	Explanation
Normal Distribution	Generates points of a given dimensionality containing numbers sampled randomly from a Gaussian distribution with a given mean and standard distribution.
Character Delimited Files	Loads a given file containing points (one per line) made up of numbers separated by a given delimiter. Capable of returning random points from the file or all of them.
Strings From File	Loads a given file containing strings (one per line) which it converts to Lists of Numbers based on the ASCII values of the characters. Capable of returning random points from the file or all of them.
SQLite Database	Pulls data from a given SQLite database using JDBC. Assumes the database contains a table which has two columns, an 'id' column which contains incrementing row numbers beginning at 1, and another column which contains data points which are numerical values separated by a given delimiter. Capable of returning random points from the database or all of them.
PCAP Files	Loads a given PCAP file and extracts the payload from each packet as a byte string. Uses these byte strings, represented as Lists of Numbers, as the points. Capable of returning random points or all of them.

In order to run a variety of model drift experiments, we implemented a MonteCarloSimulator class which requires only an Algorithm and a DataStorage. This class offered multiple functionalities which are summarized in Table 7.

Table 7: Functionalities offered by MonteCarloSimulator class in Java API

Method	Explanation	Purpose
Train and Test	For a given number of iterations: Train the algorithm using a randomly generated training set from the data storage, and then test a given test point against the resulting classifier.	Determine on average the test score of a particular point. Tests the consistency of the classifier when trained on varying training sets.
Train and Retrain	For a given number of iterations: Train the algorithm using a randomly generated training set from the data storage, and test a given test point against the resulting classifier. Then add given insertion points to the training set, retrain, and recalculate the score of the test point. Also has option to add additional random points during retraining.	Tests the model drift effects of the given insertion points.
Additive Retraining	For a given number of iterations: Train the algorithm using a randomly generated training set from the data storage, and test a given test point against the resulting classifier. Then add one of the given insertion points, retrain, and test the test point again. Iteratively repeat, adding each insertion point to the training set. Record score of test point after each insertion point is cumulatively added to the training set. Also has option to add additional random points during retraining iterations. Also has the option to calculate the 'optimal' insertion points rather than use a given list.	Tests the model drift effects of the given insertion points over time. Simulates the low and slow approach that would cause an anomalous point to gradually appear normal.
Add Optimal Until Misclassification	For the given number of iterations: Train the algorithm using a randomly generated training set from the data storage. Then calculate the 'optimal' insertion point, insert it into the training set, retrain, and test the test point. Repeat this until the test point has a score such that it would not be classified as anomalous.	Determines the amount of effort an adversary would need to put forth in order to force a misclassification on a given test point. Gives a measurement of how secure the classifier is to model drift.

The API also allowed for the simple inclusion of multiple optimization algorithms, which are used to calculate the point which induces the greatest amount of model drift in the direction of the test point when added to the training set (Figure 1). The optimization approaches detailed in section 3.2.6.

3.2.4.2 Optimization Selection were each implemented, and generally required only an Algorithm, a DataStorage, and any optimization algorithm specific parameters. Figure 12 shows a segment of the class diagram for the API and gives an overview of the class structure.

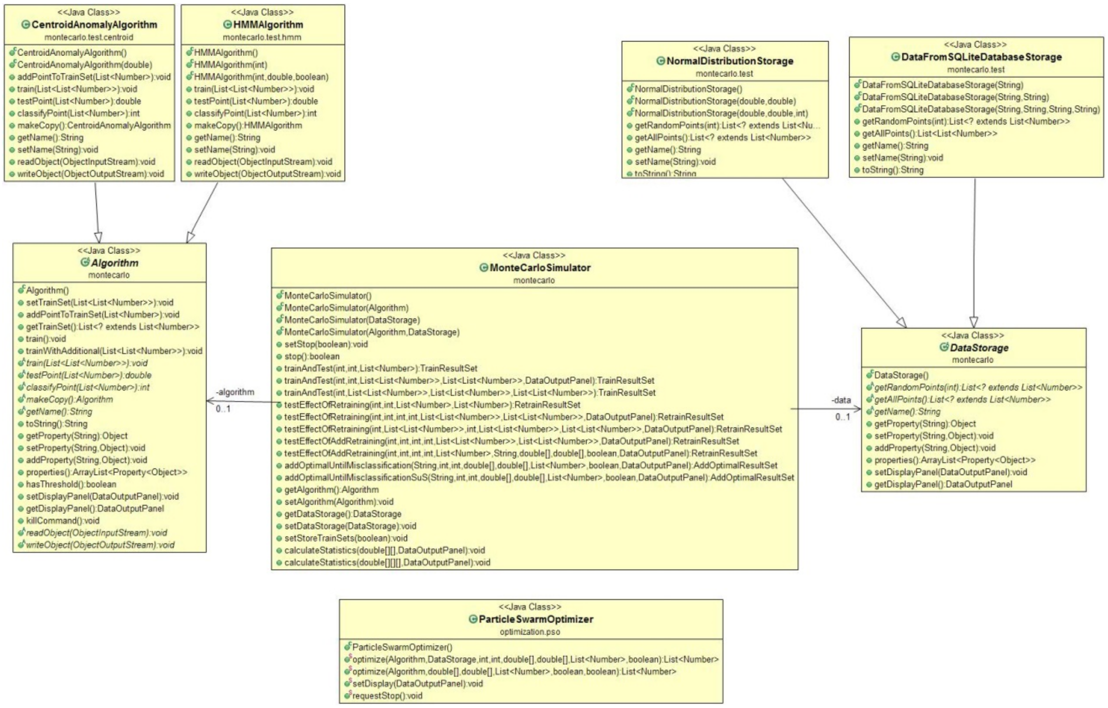
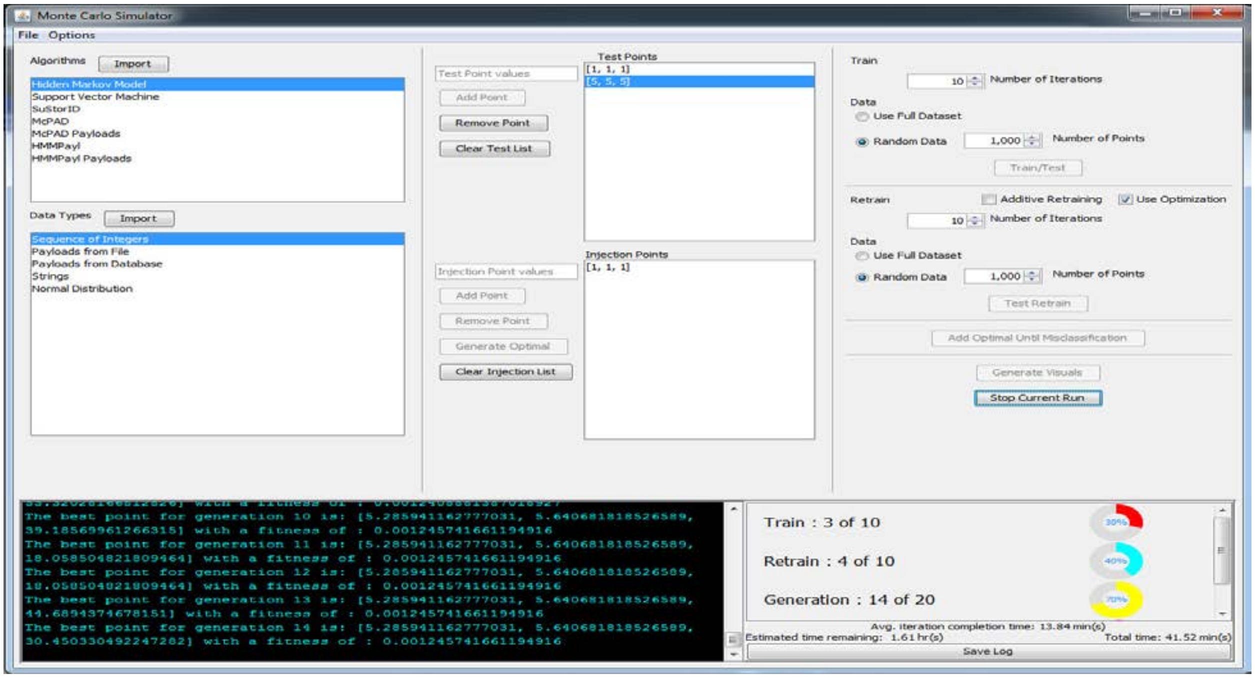

<<Java Class>>
CentroidAnomalyAlgorithm
montecarlo.test.centroid
CentroidAnomalyAlgorithm()
CentroidAnomalyAlgorithm(double)
addPointToTrainSet(List<Number>):void
train(List<List<Number>>):void
testPoint(List<Number>):double
classifyPoint(List<Number>):int
makeCopy():CentroidAnomalyAlgorithm
getName():String
setName(String):void
readObject(ObjectInputStream):void
writeObject(ObjectOutputStream):void
<<Java Class>>
HMMAlgorithm
montecarlo.test.hmm
HMMAlgorithm()
HMMAlgorithm(int)
HMMAlgorithm(int,double,boolean)
train(List<List<Number>>):void
testPoint(List<Number>):double
classifyPoint(List<Number>):int
makeCopy():HMMAlgorithm
getName():String
setName(String):void
readObject(ObjectInputStream):void
writeObject(ObjectOutputStream):void
<<Java Class>>
NormalDistributionStorage
montecarlo.test
NormalDistributionStorage()
NormalDistributionStorage(double,double)
NormalDistributionStorage(double,double,int)
getRandomPoints(int):List<? extends List<Nu...
getAllPoints():List<? extends List<Number>>
getName():String
setName(String):void
toString():String
<<Java Class>>
DataFromSQLiteDatabaseStorage
montecarlo.test
DataFromSQLiteDatabaseStorage(String)
DataFromSQLiteDatabaseStorage(String,String)
DataFromSQLiteDatabaseStorage(String,String,String,String)
getRandomPoints(int):List<? extends List<Number>>
getAllPoints():List<List<Number>>
getName():String
setName(String):void
toString():String
<<Java Class>>
Algorithm
montecarlo
Algorithm()
setTrainSet(List<List<Number>>):void
addPointToTrainSet(List<Number>):void
getTrainSet():List<? extends List<Number>>
train():void
trainWithAdditional(List<List<Number>>):void
train(List<List<Number>>):void
testPoint(List<Number>):double
classifyPoint(List<Number>):int
makeCopy():Algorithm
getName():String
toString():String
getProperty(String):Object
setProperty(String,Object):void
addProperty(String,Object):void
properties():ArrayList<Property<Object>>
hasThreshold():boolean
setDisplayPanel(DataOutputPanel):void
getDisplayPanel():DataOutputPanel
killCommand():void
readObject(ObjectInputStream):void
writeObject(ObjectOutputStream):void
-algorithm
0..1
<<Java Class>>
MonteCarloSimulator
montecarlo
MonteCarloSimulator()
MonteCarloSimulator(Algorithm)
MonteCarloSimulator(DataStorage)
MonteCarloSimulator(Algorithm,DataStorage)
setStop(boolean):void
stop():boolean
trainAndTest(int,int,List<Number>):TrainResultSet
trainAndTest(int,int,List<List<Number>>,List<List<Number>>,DataOutputPanel):TrainResultSet
trainAndTest(int,List<List<Number>>,List<List<Number>>,List<List<Number>>):TrainResultSet
testEffectOfRetraining(int,int,List<Number>,List<Number>):RetrainResultSet
testEffectOfRetraining(int,int,int,int,List<List<Number>>,List<List<Number>>,DataOutputPanel):RetrainResultSet
testEffectOfRetraining(int,List<List<Number>>,int,List<List<Number>>,List<List<Number>>,DataOutputPanel):RetrainResultSet
testEffectOfAddRetraining(int,int,int,int,List<List<Number>>,List<List<Number>>,DataOutputPanel):RetrainResultSet
testEffectOfAddRetraining(int,int,int,int,List<Number>,String,double[],double[],boolean,DataOutputPanel):RetrainResultSet
addOptimalUntilMisclassification(String,int,int,double[],double[],List<Number>,boolean,DataOutputPanel):AddOptimalResultSet
addOptimalUntilMisclassificationSuS(String,int,int,double[],double[],List<Number>,boolean,DataOutputPanel):AddOptimalResultSet
getAlgorithm():Algorithm
setAlgorithm(Algorithm):void
getDataStorage():DataStorage
setDataStorage(DataStorage):void
setStoreTrainSets(boolean):void
calculateStatistics(double[][],DataOutputPanel):void
calculateStatistics(double[][][],DataOutputPanel):void
-data
0..1
<<Java Class>>
DataStorage
montecarlo
DataStorage()
getRandomPoints(int):List<? extends List<Number>>
getAllPoints():List<? extends List<Number>>
getName():String
getProperty(String):Object
setProperty(String,Object):void
addProperty(String,Object):void
properties():ArrayList<Property<Object>>
setDisplayPanel(DataOutputPanel):void
getDisplayPanel():DataOutputPanel
<<Java Class>>
ParticleSwarmOptimizer
optimization.pso
ParticleSwarmOptimizer()
optimize(Algorithm,DataStorage,int,int,double[],double[],List<Number>,boolean):List<Number>
optimize(Algorithm,double[],double[],List<Number>,boolean,boolean):List<Number>
setDisplay(DataOutputPanel):void
requestStop():void

Figure 12: Segment of Java API class diagram

3.2.4.3 GUI. A prototype GUI was developed in Java in order to expedite experimental runs using our API. The GUI was designed to support all of the functionality of the MonteCarloSimulator class while allowing the user to quickly customize all of the experiment parameters. The GUI displays a panel which allows the user to select an implemented ML algorithm and a data source and insert any algorithm or data source specific parameters. Another panel exists which allows the user to enter or load specific test points and insertion points to use during testing. A third panel allows the user to run any of the functions listed in Table 7 and choose specific parameters such as the number of iterations to run and the size of the training set. It also has an option to bring up a graphical display of the results. A collapsible panel at the bottom displays experimental progress and textual output. The GUI supports the ability to save/load experimental setups, so that experiments can later be loaded and modified or rerun/validated. Additionally, experimental results may be saved to a file in CSV format. Figure 13 displays a screenshot of the GUI.



Algorithms: Hidden Markov Model (selected), Support Vector Machine, SuStorID, McPAD, McPAD Payloads, HMMPayl, HMMPayl Payloads

Data Types: Sequence of Integers (selected), Payloads from File, Payloads from Database, Strings, Normal Distribution

Test Point values

Test Points: [1, 1, 1], [5, 5, 5] (selected)

Injection Point values

Injection Points: [1, 1, 1]

Train: 10 Number of Iterations

Data: ○ Use Full Dataset ● Random Data 1,000 Number of Points

Retrain: □ Additive Retraining ☑ Use Optimization

10 Number of Iterations

Data: ○ Use Full Dataset ● Random Data 1,000 Number of Points

```
The best point for generation 10 is: [5.285941162777031, 5.640681818526589,
39.18569961266315] with a fitness of : 0.0012457416611949l6
The best point for generation 11 is: [5.285941162777031, 5.640681818526589,
18.058504821809464] with a fitness of : 0.001245741661194916
The best point for generation 12 is: [5.285941162777031, 5.640681818526589,
18.058504821809464] with a fitness of : 0.001245741661194916
The best point for generation 13 is: [5.285941162777031, 5.640681818526589,
44.68943746781511] with a fitness of : 0.001245741661194916
The best point for generation 14 is: [5.285941162777031, 5.640681818526589,
30.450330492247282] with a fitness of : 0.001245741661194916
```

Train : 3 of 10

Retrain : 4 of 10

Generation : 14 of 20

Avg. iteration completion time: 13.84 min(s)

Estimated time remaining: 1.61 hr(s) Total time: 41.52 min(s)

Figure 13: Screenshot of Monte Carlo Simulator GUI

Figure 14 displays an example of the graphical results provided by the GUI at the completion of an experimental run. Multiple different views of the data are available including histograms, box and whisker plots, and scatter plots which help the user quickly understand the results of the Monte Carlo iterations.

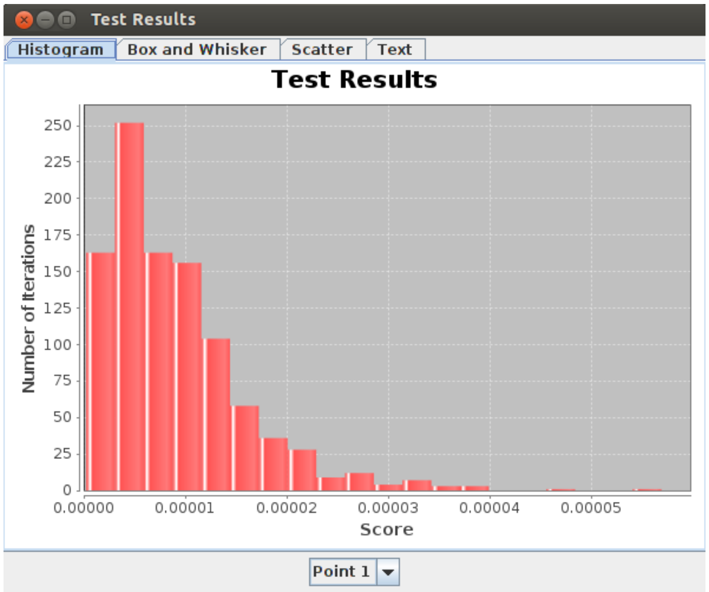


Figure 14: Screenshot of visual display of experimental results provided by Monte Carlo Simulator GUI

In the Appendices, we have included a User's Guide to aid in using the Monte Carlo Simulator GUI for running experiments. Instructions for setting up the virtual machine and accessing the simulator are in a test file included on the DVD-R with the code itself. Start by opening the file titled "README.txt." The User's Guide provides plenty of additional screen shots and information to run the Monte Carlo Simulator.

3.2.5 Model Drift Experiments. In order to test the validity of our proposed approach for analyzing the resistance of an IDS's machine learning algorithms to induced model drift, a number of experimental runs were performed. This section describes the experimental procedure used to test each of the algorithms and IDSs described in sections 3.2.1 Algorithm Selection and 3.2.2 Real World ML-IDS as well as the experimental results. We endeavored to demonstrate through repetition and careful experimentation precisely how susceptible each algorithm is to model drift induced by an adversary and to note factors that may help to reduce this weakness in each of the algorithms. As stated earlier, we focused on targeted causative integrity attacks as defined in the taxonomy developed by Barreno et al (Table 1). In this scenario, an adversary inserts specially crafted points into the training set in order to drift the learning models so that a specific point which previously appeared anomalous becomes classified as innocuous. While exploring these weaknesses in the ML models, we also sought to explicitly quantify their susceptibility.

Two sets of experiments were run using the Monte Carlo simulation framework. In the first set, the ML algorithms were used to classify color values as either normal or anomalous based on their Red/Green/Blue (RGB) values. The RGB value of a color is the extent of red, green, and blue the color contains, represented as integer values between 0 and 255 inclusive. Therefore each training point is a vector of integers of length three. The training sets used for these experiments were sampled from a Gaussian distribution with a mean of 127 and a standard deviation of 30. We then selected colors that the selected classifiers identified as anomalous, and measured the effort required by an adversary to force misclassifications using the outlined approach. While this data set is overly simplistic, it allows us to easily illustrate the above described concepts and to identify general trends. This also gave us relative ease of processing while providing sufficient variety to the random data sets produced for testing with the Monte Carlo simulations. This approach allowed us to test the algorithms isolated from their associated IDSs to more purely test the mathematical aspect of the ML algorithms separate from any potential implementation issues that may exist in the IDSs.

The second set of experiments was designed to be more realistic, testing the resiliency of the actual IDS implementations to adversarial drift using network data. In initial experiments, we generated our own generic network traffic of the sort required for the IDSs covered. However, for the majority of the experimental runs, we utilized the DARPA'99 dataset to train the anomaly detection models. Although the DARPA'99 dataset is outdated and has been widely criticized, it is still highly appropriate and valuable for our needs. It is the most common public dataset used to baseline ML-IDS and lends to the repeatability of our experiments. Additionally, the largest complaint against DARPA'99 is that it is no longer suitable for measuring the accuracy of an IDS and its ability to detect intrusions. However, this is not the context for which we were using it. Our goal was to find appropriately formatted network data and some attack point that is flagged as anomalous by the classifier trained on this data. This approach does not consider the overall accuracy of the classifier, but rather its ability to continue making an accurate prediction despite adversarial attempts to induce model drift. Initially, we used a real HTTP attack dataset provided by the authors of McPAD on their project website as our test points on which to force misclassifications. We later inspected an alternate attack dataset (available at https://www.mediafire.com/?a49l965nlayad#7vz9n6749t1ej) which offered more recent and a wider variety of attacks.

3.2.5.1 Color Experiments. In order to illustrate the above concepts and concerns, we conducted experiments utilizing an anomaly detector designed to classify color values as either normal or anomalous based on their RGB values. Each training point's red, green, and blue values were sampled from a Gaussian distribution with a mean of 127 and a standard deviation of 30. Figure 15 displays 2000 colors randomly sampled from this distribution, plotted on the Cartesian axes. The X axis represents the red value, the Y axis represents the green value, and the Z axis represents the blue value.

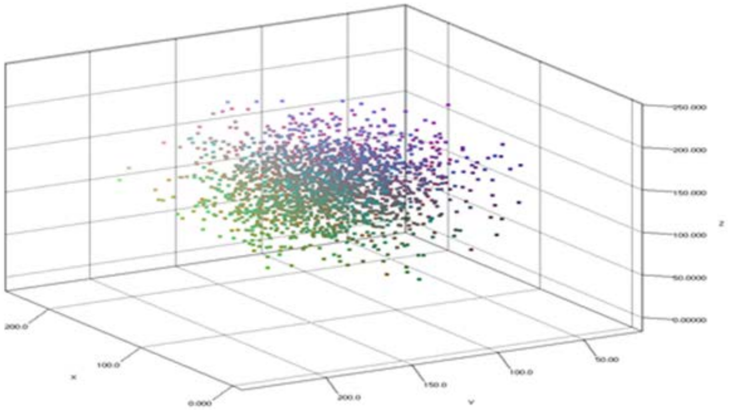


Figure 15: Colors randomly sampled from a Gaussian distribution

A number of experiments were run using the color RGB data and the identified anomaly detection ML algorithms. For initial tests, we selected two colors that were consistently classified as anomalous by each of the algorithms. The colors were green-yellow and goldenrod which had RGB values of [173, 255, 47] and [255,193,37] respectively. Then, for each algorithm, we determined the effort, as defined in Figure 1, required by an adversary to force a misclassification on the two selected target points. This was accomplished by selecting a random training set, training the algorithm, and then iteratively generating an insertion point, adding it to the training set, retraining, and measuring the new test score of the target point until the target is no longer classified as anomalous. The number of insertion points required to force the misclassification is recorded and defined as the effort required by the adversary. We varied the size of the training set and for each training set size ran multiple iterations with randomized data. In initial experiments, we investigated HMMs and used Particle Swarm Optimization to generate the insertion points. The results of these experiments can be found in our paper written for SPIE DSS entitled "Evaluating data distribution and drift vulnerabilities of machine learning algorithms in secure and adversarial environments," which is included in the appendix.

During this experiment, we also evaluated the correlation between training set size and the adversary's ability to alter an anomalous point's probability with a single insertion point. For each training set, the HMM was trained and the anomalous point's probability score was determined. Then, an insertion point was generated using PSO, the point was added to the training set, the

HMM was retrained, the anomalous point's new probability score was computed, and the difference between the two probability scores was calculated. The effect that an adversary can have on the probability score of an anomalous point using a single insertion point had an almost perfect inverse relationship with the number of points in the training set. This negative correlation implies that increasing the size of the training set may mitigate some of the risk of a targeted causative attack against an ML algorithm. However, the time and resource cost of training an ML algorithm increases as the number of training points increases, and at some point, this cost must be weighed against the potential impact of an adversary. The results of this experiment are also located in our paper written for SPIE DSS.

In later experiments, we investigated the effort required by an adversary to force model drift for each of the identified ML algorithms using the algorithm-specific optimization methods described in section 3.2.8.

3.2.5.2 Optimization Selection. We varied the size of the training set and for each training set size ran multiple iterations with randomized data. The results of these experiments are summarized in Figure 16. The y-axis on the plots has been scaled logarithmically due to the large disparity between the algorithms. However, it should be noted that the relationship between the training set size and effort required by the adversary was in fact linear. These experiments allow for a simple comparison between algorithms. For the purpose of detecting anomalous colors, SVMs appear to be significantly more susceptible to adversarial drift than a simple centroid anomaly detector. This also shows the advantage of using a large training dataset for defending against adversarial drift, which must be weighed against the increased cost of acquiring data and training. More information about these experiments can be found in our paper written for IEEE CISDA entitled "Evaluation Model Drift in Machine Learning Algorithm," which is included in the appendix.

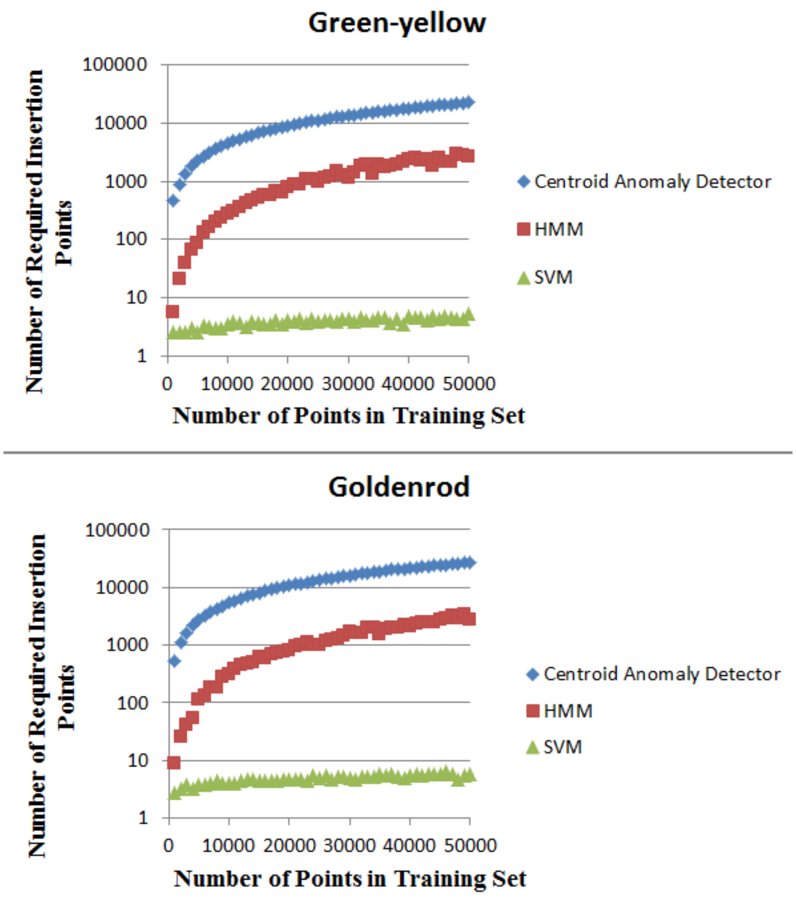


Figure 16: Effort required by an adversary to cause misclassifications of selected anomalous color values using various ML algorithms

In the next experiment, for a fixed training set size, we tested each algorithm with a wide variety of different colors to determine the number of points necessary for the adversary to insert in order to force a misclassification on each. This value was then compared against the initial test scores to gain a better understanding of the relationship between the extent to which a point is anomalous and the model's resistance to adversarial drift towards the point. The results of this experiment are summarized in our paper written for IEEE CISDA. The centroid anomaly detector and HMM both show a clear correlation between anomaly score and effort required by the adversary. The centroid anomaly detector shows a positive correlation because its test score represents a distance from normalcy, while HMM shows a negative correlation because its test score represents a probability of being benign. The relationship for the SVM is not as clear due to many of the points' initial test scores rounding to zero, but there appears to be a loosely negative correlation.

Additionally, we ran experiments using varying configurations of the centroid anomaly detector. The results are detailed in the IEEE CISDA paper. These results give a clear indication of which configuration causes the most effort for the adversary and is therefore the most secure against induced model drift.

In further experiments, our goal was to compare the performance and effectiveness of the various optimization algorithms developed for generating insertion points. We ran experimental iterations to test the various algorithms discussed in section 3.2.1 Algorithm Selection and the optimization approaches discussed in section 3.2.9.

3.2.5.3 Optimization Selection. We selected a training set of 1000 three-dimension points randomly selected from a normal distribution with mean 127 and a standard distribution of 30 to use for each experimental iteration. We also chose to use green-yellow [173, 255, 47] as our anomalous test point. Then, for each ML algorithm and each optimal point generation approach, we calculated the number of innocuous insertion points required to successfully drift the ML models to allow the test point and the amount of time required for the calculations. For the genetic algorithm and the genetic algorithm with AFFG, we used a mutation rate of 5%, a population size of 50, and an iteration count of 500. For the Nelder-Mead optimizer, we chose to run the algorithm for 50 iterations, and the simulated annealing algorithm ran for a maximum of 1000 iterations. The results of these experiments are summarized in Table 8.

Table 8: Performance comparison of optimal insertion point generation approaches

Centroid Anomaly Detector			
Optimization Algorithm	Number of Required Insertion Points	Time (Seconds)	Time Per Point (Seconds)
Centroid Anomaly Optimizer	545	0.546	0.001001835
Genetic Algorithm	683	11474.975	16.80084187
AFFG Genetic Algorithm	604	9759.453	16.15803477
Nelder-Mead	839	23.085	0.027514899
Simulated Annealing	578	590.008	1.020775087
Hidden Markov Model			
Optimization Algorithm	Number of Required Insertion Points	Time (Seconds)	Time Per Point (Seconds)
Centroid Anomaly Optimizer	6	0.297	0.0495
Genetic Algorithm	6	2551.304	425.2173333
AFFG Genetic Algorithm	6	292.221	48.7035
Nelder-Mead	7	25.406	3.629428571
Simulated Annealing	10	1552.281	155.2281
K-Means Anomaly Detector			
Optimization Algorithm	Number of Required Insertion Points	Time (Seconds)	Time Per Point (Seconds)
K-Means Anomaly Optimizer	13	0.561	0.043153846
Genetic Algorithm	161	30549.549	189.7487516
AFFG Genetic Algorithm	360	116828.75	324.5243056
Nelder-Mead	180	1058.763	5.882016667
Simulated Annealing	13	981.145	75.47269231

3.2.6 PCap Experiments. The primary purpose of these initial color RGB experiments was to test the legitimacy of the framework's capabilities and to discover baseline patterns. The next step was to validate these results using the identified open-source IDSs with actual network data.

We began initial experiments with SuStorID in an effort to induce drift on its HMM models. We utilized the technologies listed in section 3.2.9.1.1.

3.2.6.1 Technologies Used to automatically generate requests to our simple test web application that was protected by SuStorID. For our initial test point, we selected a value that when entered into the text field on the sample HTML page would be classified as an anomaly by SuStorID's models trained with our generated traffic. Although this test point did not represent a genuine attack against the web app, it suited our purpose for initial testing as it was an anomaly that could be used as the target for model drift. While we were able to demonstrate drift manually during testing, it soon became apparent that SuStorID itself was much too unreliable for automated testing. The system would often crash and need to be restarted during the training phase. This made it very ill-suited for automated iterative testing which relied upon repetitive retraining. For this reason, few results are available and we instead chose to focus on a different IDS which utilizes HMMs, HMMPayl.

HMMPayl required training data from PCap files in order to create its learning models. As mentioned above, we chose to use the DARPA '99 data set as our source of training data. In our literature review, we found the DARPA '99 data set to be a very common data set used by researchers to develop and test the their IDSs and thus was a convenient source of network traffic to use in our own testing. It allowed us to verify their results to an extent and to compare our results to theirs in part as well. For our test points, we chose to use the attack data provided by the authors of McPAD on their project website. The data set consisted of a number of PCap files that each represented an HTTP attack. This provided us with genuine attacks that we could test the IDS against. For an initial test, we selected an intrusion point from the identified attack data set that was consistently flagged by HMMPayl when trained with data from the DARPA'99 dataset. This attack point actually consisted of seven packet payloads representing a chunked encoding transfer heap overflow against Microsoft IIS. The HMM-specific approach for generating insertion points was applied until each of the seven payloads went undetected by the IDS. This was repeated multiple times with randomly selected normal traffic for varying training set sizes. The results of this experiment can be found in our paper written for IEEE CISDA. A linear relationship between the training set size and the number of insertion points required by the adversary became apparent. The slope of the best-fit line revealed that on average the adversary need only insert 0.486% of the training set size to successfully induce model drift while remaining undetected.

Similar to the color experiments, we next selected every individual packet payload from the attack dataset. For a fixed training set size, we determined the number of insertion points required by the adversary in order to create a misclassification on each payload. We again used the HMM-specific approach for generating insertion points. The required number of points is compared against the initial test score of the payloads to give the security administrator an overall feel for the resiliency of the system. The results of this experiment are summarized in our paper written for IEEE CISDA. For the selected attack points there is a loosely negative correlation between the initial test score and the required number of insertion points. It can also be seen that the adversary needs to insert no more than 0.1% of the training set in order to create a misclassification on a single payload.

A more careful examination of the attack data set revealed that the payloads detected by the IDS were very homogenous. Many of the payloads were similar to each other and consisted primarily of the same byte pattern, two bytes repeating iteratively. For this reason we identified an additional attack data set which contained more recent data and offered a greater variety of payloads. Several PCap files were downloaded from this source (available at https://www.mediafire.com/?a49l965nlayad#7vz9n6749t1ej) and were used as our test data set. For a fixed training set size of 500, we determined the number of insertion points required by the adversary in order to create a misclassification on each payload from the new test data set of attacks. We again used the HMM-specific approach for generating optimal points. Figure 17 summarizes the results of this experiment. At a maximum, 254 insertion points, or 50.8% of the training set, were required. However, for a number of payloads, only 1 insertion point was required to cause the misclassification. On average, about 36.5 insertion points (7.3% of the training set) were required for each payload, with a median value of 21 insertion points (4.2% of the training set).

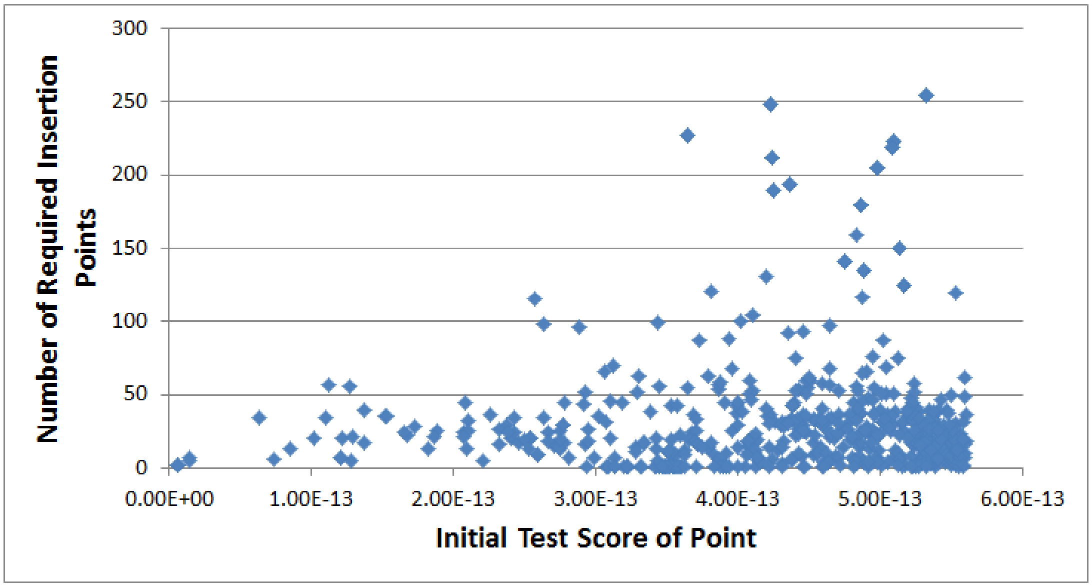


Figure 17: Effort required by an adversary to cause HMMPayl to misclassify selected attack payloads

We additionally attempted to run similar experiments using McPAD as our IDS in an effort to drift its SVM models. It soon became apparent, however, that the SVM-specific approach developed for generating optimal points that was successful during the RGB experiments did not extrapolate well to non-Gaussian network data. Using this method, we were unable to induce the desired drift effects. We therefore attempted each of the universal optimization methods described in section 3.2.3.5 ***Universal Optimization***. Again, we were met with little success. Often we were able to increase the score of the test point, but the threshold would also increase at an equal or greater rate, causing the point to remain anomalous. Unfortunately, this was not resolved, and it appears as though the methods utilized by McPAD are relatively secure to adversarial drift. Even when removing McPAD's advantage of using an ensemble and instead using a single SVM model, we had limited success.

3.2.7 HPC Experiment. As mentioned above, one of the major drawbacks of our chosen approach for evaluating ML systems and for generating 'optimal' insertion points is the computational time requirement. Our method for measuring the security of an ML system against model drift requires iteratively adding points to the training set and retraining. This retraining of the learning models may be an expensive process, especially for larger data sets. Additionally, the fitness function of our heuristic-based optimization approaches for generating insertion points also requires retraining the ML models. In an effort to mitigate this limitation and to greatly expedite our experimental runs, we acquired access to the High Performance Computer (HPC) Condor cluster available to AFRL researchers.

In order to take advantage of the large amount of processing power, available memory, and multiple nodes on the HPC, a distributed version of the Monte Carlo Simulator Java API was developed. A class was created which was responsible for tasks such as training the ML algorithm models, testing given test points, and generating 'optimal' insertion points. Then, a server class was created which creates an RMI registry and binds an instance of the other class to the registry. This server would be started and run on each of the available nodes on the HPC, waiting for requests to train/test an algorithm or generate optimal points. The MonteCarloSimulator class was then altered so that iterations of the current task would be run simultaneously on the remote servers, communicating through remote method invocation (RMI). Each of the functionalities listed in Table 7 was modified to be able to run in a distributed manner as such. Additionally, to take advantage of the processing power and memory of each node, the methods to be carried out on the servers were also multi-threaded, so that multiple iterations could be run concurrently on each. This meant that *n*t* iterations could be run simultaneously, where *n* is the number of available nodes, and *t* is the number of threads to run on each. This obviously decreased the time requirement for experimental runs.

Additionally, effort was made to increase the speed at which optimal points are generated through heuristic algorithm-independent methods. We chose to work with genetic algorithms as they initially appeared to produce the best results when given time to run to completion. Also, genetic algorithms are easily distributable. The fitness function calculation is the most expensive process and occurs for each member of the population during each generation. However, this is an independent process, so the fitness values of all members of a given population may be calculated concurrently. To implement this, we developed a class which is responsible only for calculating fitness values. Then, a server class was developed which creates an RMI registry and binds an instance of the fitness calculation class to the registry. This server would be started and run on each of the available nodes on the HPC, waiting for requests to calculate the fitness value for a given test point. We then altered the genetic optimizer class so that fitness calculations for the members of the current population are run simultaneously on the remote servers, communicating through RMI. The fitness calculation class also allowed for multi-threading, enabling multiple fitness values to be calculated concurrently on a single node, taking advantage of the processing power and memory on the nodes. This meant that the fitness of *n*t* points could be calculated simultaneously during each generation. To further speed up calculations, the Adaptive Fuzzy Fitness Granulation method described in section 3.2.3.5.3 was implemented into the distributed genetic algorithm.

A number of experiments were run utilizing the distributed implementation of the Monte Carlo Simulator on the HPC. In order to compare the performance of the genetic algorithm on the HPC versus on a standalone machine, a simple test was run to generate optimal points that would drift an HMM to accept an anomalous point. Ten-dimensional Gaussian training data was used with a training set size of 300 points, and the genetic algorithm used a population size of 50 and ran for 500 iterations. When running 5 threads concurrently on a standalone laptop with a quad-core 2.70 GHz processor and 16.0 GB of RAM, it took about 397 seconds on average to generate each optimal insertion point. When running on the HPC with 7 nodes and 5 threads running concurrently on each, it took about 116 seconds on average to generate each optimal insertion point. With just 7 nodes, the HPC allowed points to be generated over three times as quickly.

A similar experiment was attempted using HMMPayl and network training data from the DARPA ’99 data set. However, using just 200 packets for training and a population size of 100 for 100 generations, the genetic algorithm took roughly 26 hours to generate just 5 insertion points. This was deemed unreasonably long, and alternate optimization methods were investigated at this point.

Although the HPC did not improve the speed of the algorithm-independent optimization methods to quite the extent that we had initially hoped, it still proved to be useful for decreasing the time requirements for experimental runs. Specifically, when running tests using the faster algorithm-specific optimization methods, many points could be tested simultaneously. Also, more iterations could be run as each one processed more quickly, leading to less variance in results.

3.2.8 ICS and SCADA. Having been granted access to Industrial Control System (ICS) and Supervisory Control And Data Acquisition (SCADA) specific networking hardware, we produced an experiment plan for proposed work using our methodology in the context of studying and assessing ICS and SCADA security systems which use learning systems to handle any portion of the network threat identification. Details are available at a higher security level and are not included in this document.

4.0 CONCLUSIONS

As a result of our research, we have created a methodology to explore the susceptibility of algorithms used in research-based ML-IDS to induced data drift while they are operating in an adversarial environment. The methodology was developed while examining and subsequently testing several anomaly detectors to establish the baseline approach and results. We further developed and validated the methodology through analysis of additional algorithms implemented in an ML-IDS. We identified potential heuristics to create insertion points in order to induce data drift. We then ran a series of experiments to thoroughly exercise the identified ML-IDSs in order to explore their susceptibility to induced data drift while operating in our tightly controlled adversarial environment. We progressed from the overly-simplified RGB values used to establish baseline results to using real-world network traffic data. Our initial experiments demonstrate the type of valuable information that a system administrator may gain through the use of our framework, and preliminary results indicate that the ML algorithms utilized by ML-IDS are indeed susceptible to induced data drift while operating in an adversarial environment.

4.1 Recommendations

While developing future ML-based systems that are to be deployed in adversarial environments, it is essential that the security of the learning models is considered. We have demonstrated that in many cases it is a trivial matter for an adversary to force a misclassification of an intrusive point simply through the addition of innocuous points into the training corpus. For this reason, in addition to functional and integration testing, it is vital that the susceptibility of these systems to induced data drift is also thoroughly tested in order to ensure that the systems are as hardened as possible to vulnerabilities. Our framework and methodology provide an ample starting point for testing and measuring the security of ML models.

The suitability of a ML algorithm is often determined by calculating the accuracy, precision, or recall during a validation phase. Researchers typically make the assumption that data distributions will remain the same at test time as during validation. However, this stationary assumption is often violated, particularly in dynamic environments such as network security. This issue is only exasperated by the presence of an adversary. In order to mitigate this, researchers must develop their ML algorithms and models with security in mind from the offset. This includes choosing methods that are robust to noise and avoid overfitting to outliers. This limits the detrimental effects that an adversary may inflict through the addition of insertion points and additionally allows the model to generalize better to new data. Researchers should also consider ensemble methods, or using the result of multiple learning algorithms combined to make a prediction. This has been shown to not only improve overall predictive performance, but would also increase the amount of work required by the adversary as they would be forced to induce drift on multiple models.

Developers of ML-based systems should also be sure to limit the amount of information available to an adversary. This includes ensuring that information about the training set is secure so the adversary is not able to determine data distributions or other information that would allow them to infer details of the ML models. Also, information about the type of algorithms and models utilized should be kept secret as it would better allow the adversary to develop a targeted drift strategy. During run-time, the resulting score of testing points against the trained models should not be made available to the adversary. The test score shows them the exact progress and effectiveness of their drift attack, enabling them to further tailor and improve their strategy.

As mentioned above, ML-IDS operate in a dynamic environment in which the relevant factors and standards of normalcy are constantly changing. Therefore, periodic retraining of the anomaly detection models will remain an important aspect to ensure that the models remain current and effective. However, we have shown in our research that this retraining process creates an opportunity for an adversary to insert traffic and negatively impact the learning models. Developers must take precautions to limit the effects of the adversary during these necessary retrain periods. Retraining should not be done at regular intervals as this period may easily become known to the adversary. Additionally, not every point received should be included in the new training set to update the models. An effective method may be to randomly select points from a pool of potential points. The adversary would then have no guarantee that their insertion points are included, and would increase their required effort. Also, points considered for retraining should be tested against the existing models to ensure that points classified as obviously anomalous or malicious are not included.

Although we have observed that ML-based systems are indeed susceptible to induced model drift, it is still our belief that ML-IDS is a promising field of research and should continue to be explored. Traditional signature-based IDSs generally fail to detect zero-day and polymorphic attacks. ML-IDSs aim to solve this issue by detecting general patterns indicative of an attack rather than specific signatures, and research has shown a considerable amount of success in this regard. Current ML-IDSs are not perfect though, and are not yet widely deployed, leaving much room for research and improvement. As the field matures, however, we believe that these ML approaches will become pervasive and important aspects of cyber security. Specifically, we have observed that anomaly detection algorithms are more commonly developed and deployed due to their lack of reliance on labeled training data, which is often difficult and expensive to obtain.

We also recommend that any ML-IDS is deployed in tandem with a signature-based IDS. While signature-based IDSs are ineffective at detecting unknown attacks, they are still very valuable for detecting attacks for which a known signature exists. Due to the generalization of the models created by ML-IDS, there will inevitably be a non-negligible number of false negatives. These unflagged attacks may often be caught by a signature detection method. Additionally, this makes the system more secure as those attacks with a known signature are impervious to the effects of model drift.

When generating insertion points to test a ML-based system's resistance to induced model drift, we recommend using a specialized algorithm-specific method over a universal optimization approach. Our algorithm-specific approaches were much faster as they did not require repeated retraining of the ML models. This becomes particularly relevant as the training set size increases. They also tended to be more effective, as less points generated using this method were required to force misclassifications. The drawback to this approach, of course, is that a prior understanding of the ML algorithms is required, which may not always be available. In this case, the universal approaches are relatively effective, but involve a much greater time commitment.

4.2 Future Research

There are many interesting and novel directions in which the research may progress from this point beyond just further data collection and analysis. Future research should extend to include:

- Exploring algorithms not yet covered which have been used or may be used by other ML-IDS.
- Testing alternative libraries that implement included/excluded algorithms to explore sensitivities related to implementations across identical algorithms.
- Investigating additional data types that are considered by IDSs, both commercial and in research, such as trace and log file parsers, executable analyzers, and even multi-session analyzers.
- Exploring optimization methods to create an improved method for generating insertion points that is universal while also being feasible time and computational power-wise.
- Investigating additional sections of the ML attack taxonomy including evasion attacks which do not require influence over the training set.
- Studying defensive remediation that can be used to mitigate the vulnerabilities observed as a part of this work.
- Investigating ML-based IDS in ICS/SCADA networks to provide suggestions to make these systems more resilient and robust.
- Creating a Metasploit module for our methodology to be used by our cyber forces to be used for pen testing Department of Defense resources to further harden our networks for improved network defense.
- Abstracting the research to create a general methodology for measuring the security and suitability of ML algorithms in a domain-independent context.

Additionally, future research should investigate the drift susceptibility of the above systems after reducing the assumptions and prior knowledge/access granted to an adversary. As mentioned, this research assumed a worst-case scenario from the defender's perspective and limiting the assumptions would give a more realistic picture of the true security of the system. This includes removing the adversary's knowledge of the ML algorithm implemented, the decision boundaries, and the training set, which would then require additional careful probing by the adversary in order to ascertain this information and maintain con.

Mitigating the identified weaknesses of the ML systems is an important topic of future research. Barreno et al. describe various defensive measures which may be utilized to harden an ML system. These techniques include:

- Reject on negative impact (RONI) defense – measures effects of each training instance and rejects points which are seen to have a negative impact on classification.
- Robust algorithms – based upon Robust Statistics. The goal is to create a procedure which will limit the impact of deviant points by accounting for qualitative robustness, the breakdown point, and the influence function of the procedure.
- Online learning with experts – uses a set of classifiers each designed to provide a different security property and predictions/advice for training.
- Hide training data – if access to the training data is denied, an adversary is unable to determine the exact decision boundaries of the models used by the ML so as to analyze a way to bypass them.
- Good feature selection – make classifiers difficult to reverse engineer through careful selection of features which are kept secret and possibly even mapping raw features into a different feature space altogether.
- Limited/misleading feedback – provide feedback to attacker that provides as little information as possible revealing their level/lack of success during the probing attack.

Future research should take into account and analyze these defensive measures and include an investigation into the efficacy of the defensive measures when applied to specific ML algorithms. Utilizing the analysis method described in the above experiments, future experiments may be run in order to paint a better picture of the defensive capabilities of ML algorithms in adversarial environments.

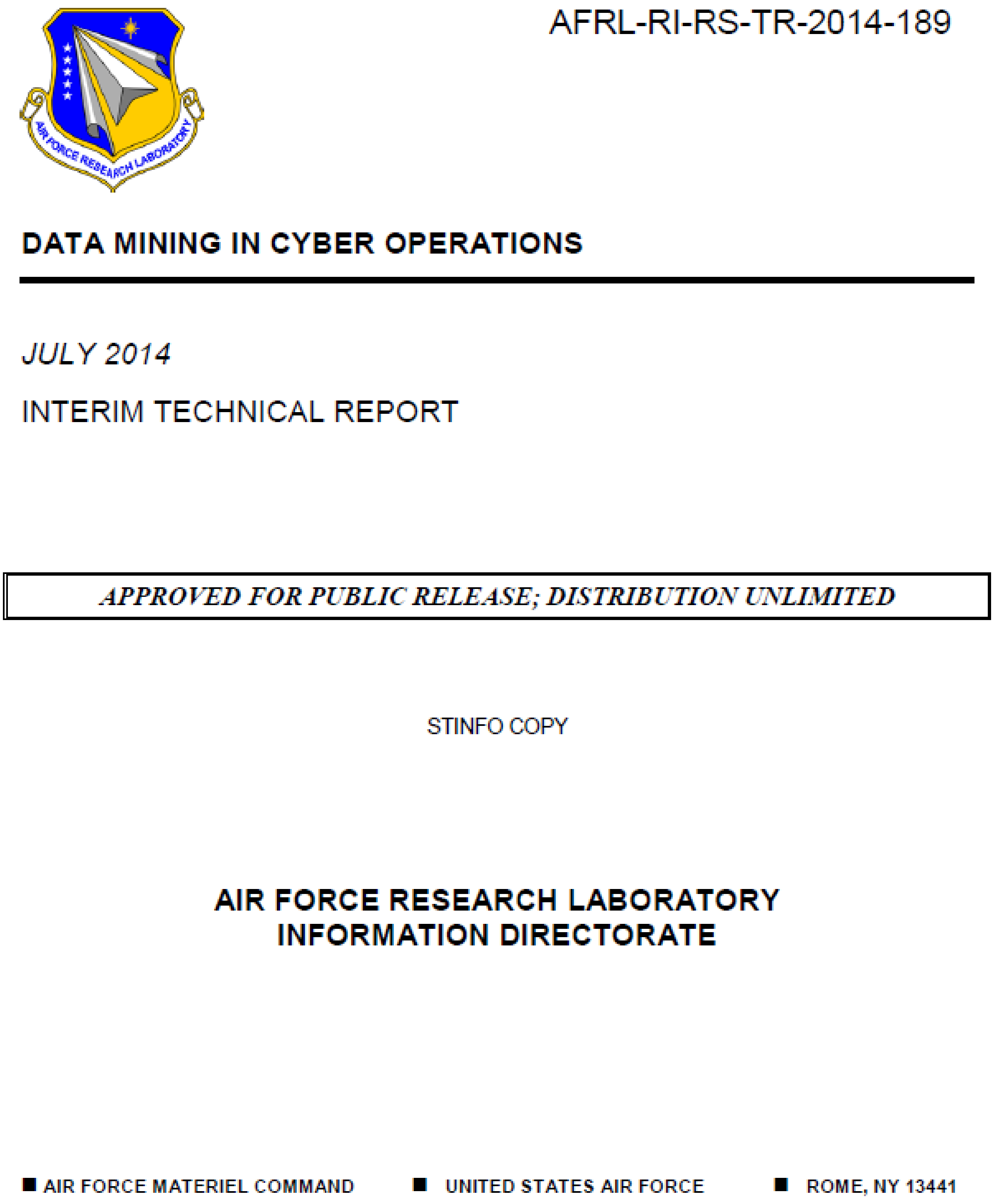
5.0 REFERENCES

M. Barreno, B. A. Nelson, A. D. Joseph, and J. Tygar, “The security of machine learning,” in Machine Learning, vol. 81, no. 2, 2010, pp. 121-148.

C. Tsai, Y. Hsu, C. Lin, W. Lin, "Intrusion detection by machine learning: a review", in Expert Systems with Applications, Vol. 36, Elsevier, 2009, pp. 11994–12000.

Tavallaee, M., Stakhanova, N., & Ghorbani, A. A. (2010, September). Toward Credible Evaluation of Anomaly-Based Intrusion-Detection Methods. *IEEE Transactions on Systems, Man, and Cybernetics - Part C, 40*(5), 516-524.

APPENDIX A - Data Mining in Cyber Operations (Cybersecurity Systems for Human Cognition Augmentation)



AFRL-RI-RS-TR-2014-189

DATA MINING IN CYBER OPERATIONS

JULY 2014

INTERIM TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

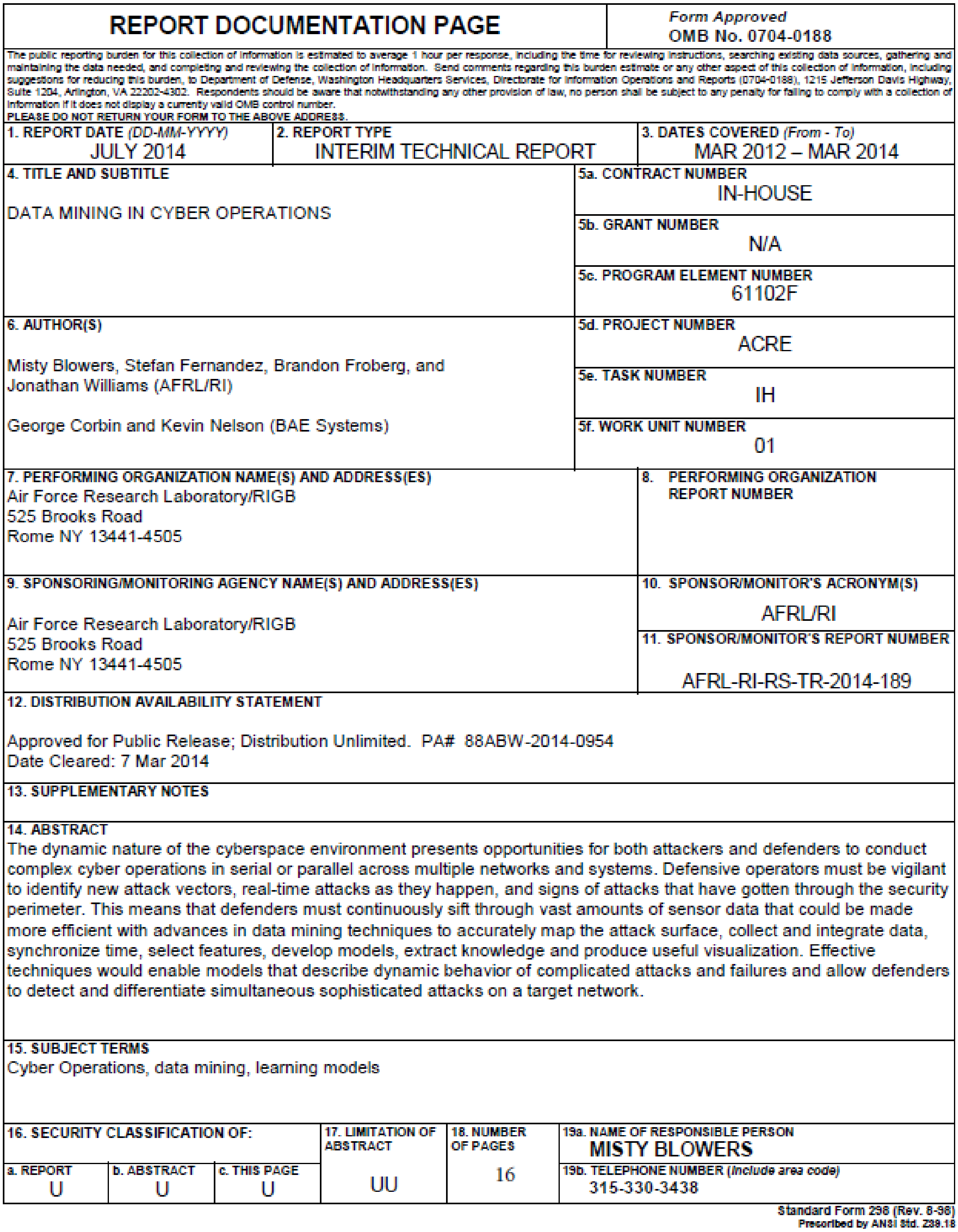
■ AIR FORCE MATERIEL COMMAND ■ UNITED STATES AIR FORCE ■ ROME, NY 13441

NOTICE AND SIGNATURE PAGE

AFRL-RI-RS-TR-2014-189 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/ S /	/ S /
MISTY BLOWERS	BRENT HOLMES
Work Unit Manager	Chief, Cyber Operations Branch
	Information Directorate



REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)* JULY 2014	**2. REPORT TYPE** INTERIM TECHNICAL REPORT	**3. DATES COVERED** *(From - To)* MAR 2012 – MAR 2014

4. TITLE AND SUBTITLE

DATA MINING IN CYBER OPERATIONS

5a. CONTRACT NUMBER
IN-HOUSE

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
61102F

6. AUTHOR(S)

Misty Blowers, Stefan Fernandez, Brandon Froberg, and Jonathan Williams (AFRL/RI)

George Corbin and Kevin Nelson (BAE Systems)

5d. PROJECT NUMBER
ACRE

5e. TASK NUMBER
IH

5f. WORK UNIT NUMBER
01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/RIGB
525 Brooks Road
Rome NY 13441-4505

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RIGB
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2014-189

12. DISTRIBUTION AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. PA# 88ABW-2014-0954
Date Cleared: 7 Mar 2014

13. SUPPLEMENTARY NOTES

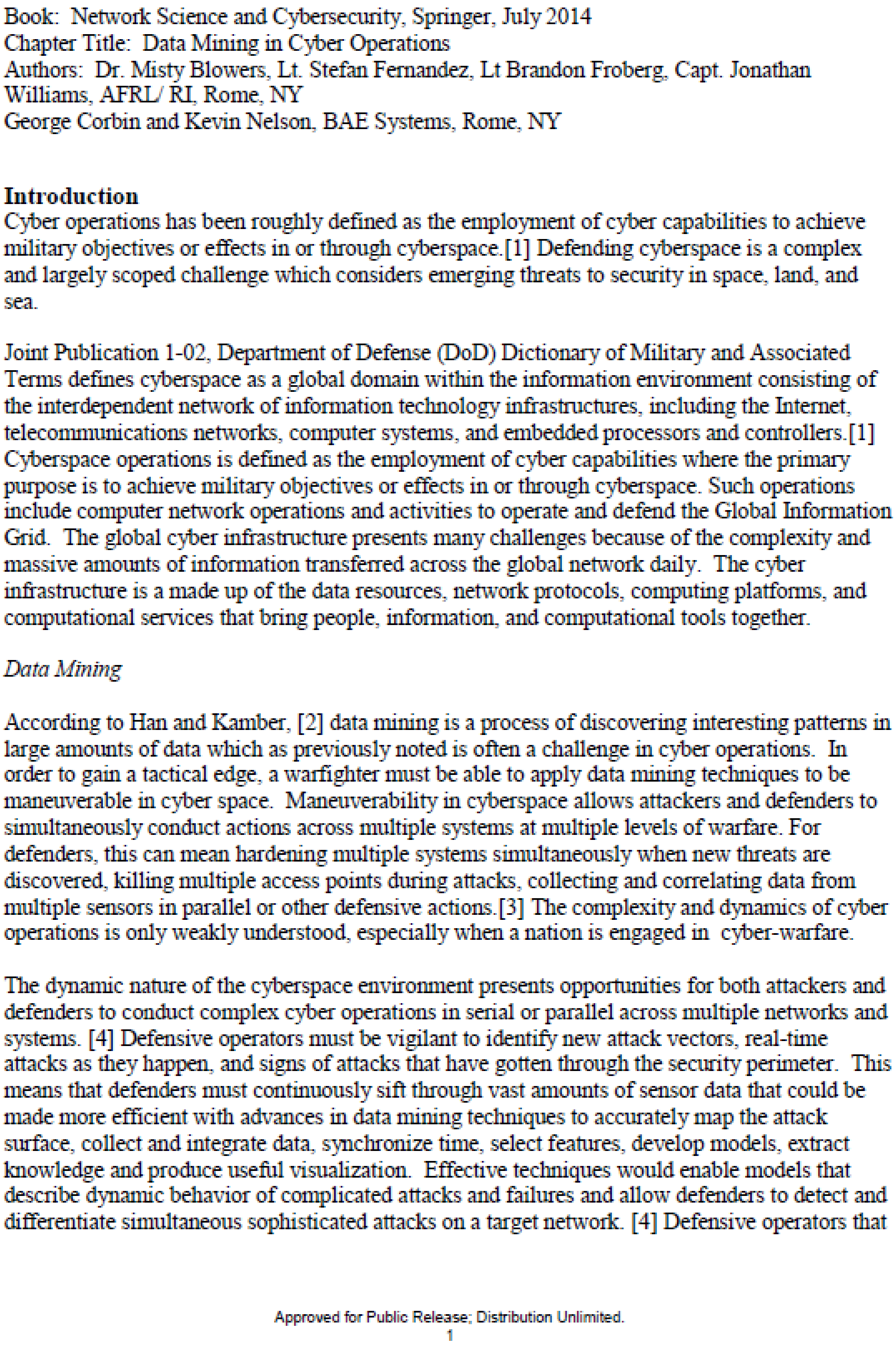
14. ABSTRACT

The dynamic nature of the cyberspace environment presents opportunities for both attackers and defenders to conduct complex cyber operations in serial or parallel across multiple networks and systems. Defensive operators must be vigilant to identify new attack vectors, real-time attacks as they happen, and signs of attacks that have gotten through the security perimeter. This means that defenders must continuously sift through vast amounts of sensor data that could be made more efficient with advances in data mining techniques to accurately map the attack surface, collect and integrate data, synchronize time, select features, develop models, extract knowledge and produce useful visualization. Effective techniques would enable models that describe dynamic behavior of complicated attacks and failures and allow defenders to detect and differentiate simultaneous sophisticated attacks on a target network.

15. SUBJECT TERMS
Cyber Operations, data mining, learning models

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** MISTY BLOWERS
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	16	**19b. TELEPHONE NUMBER** *(Include area code)* 315-330-3438

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18



Book: Network Science and Cybersecurity, Springer, July 2014
Chapter Title: Data Mining in Cyber Operations

Authors: Dr. Misty Blowers, Lt. Stefan Fernandez, Lt Brandon Froberg, Capt. Jonathan Williams, AFRL/ RI, Rome, NY
George Corbin and Kevin Nelson, BAE Systems, Rome, NY

Introduction

Cyber operations has been roughly defined as the employment of cyber capabilities to achieve military objectives or effects in or through cyberspace.[1] Defending cyberspace is a complex and largely scoped challenge which considers emerging threats to security in space, land, and sea.

Joint Publication 1-02, Department of Defense (DoD) Dictionary of Military and Associated Terms defines cyberspace as a global domain within the information environment consisting of the interdependent network of information technology infrastructures, including the Internet, telecommunications networks, computer systems, and embedded processors and controllers.[1] Cyberspace operations is defined as the employment of cyber capabilities where the primary purpose is to achieve military objectives or effects in or through cyberspace. Such operations include computer network operations and activities to operate and defend the Global Information Grid. The global cyber infrastructure presents many challenges because of the complexity and massive amounts of information transferred across the global network daily. The cyber infrastructure is a made up of the data resources, network protocols, computing platforms, and computational services that bring people, information, and computational tools together.

Data Mining

According to Han and Kamber, [2] data mining is a process of discovering interesting patterns in large amounts of data which as previously noted is often a challenge in cyber operations. In order to gain a tactical edge, a warfighter must be able to apply data mining techniques to be maneuverable in cyber space. Maneuverability in cyberspace allows attackers and defenders to simultaneously conduct actions across multiple systems at multiple levels of warfare. For defenders, this can mean hardening multiple systems simultaneously when new threats are discovered, killing multiple access points during attacks, collecting and correlating data from multiple sensors in parallel or other defensive actions.[3] The complexity and dynamics of cyber operations is only weakly understood, especially when a nation is engaged in cyber-warfare.

The dynamic nature of the cyberspace environment presents opportunities for both attackers and defenders to conduct complex cyber operations in serial or parallel across multiple networks and systems. [4] Defensive operators must be vigilant to identify new attack vectors, real-time attacks as they happen, and signs of attacks that have gotten through the security perimeter. This means that defenders must continuously sift through vast amounts of sensor data that could be made more efficient with advances in data mining techniques to accurately map the attack surface, collect and integrate data, synchronize time, select features, develop models, extract knowledge and produce useful visualization. Effective techniques would enable models that describe dynamic behavior of complicated attacks and failures and allow defenders to detect and differentiate simultaneous sophisticated attacks on a target network. [4] Defensive operators that manage an enterprise-level network, distributed networks or multiple, interoperating networks face a significant challenge of strategic coordination to defend against complex cyber-attacks. These operators clearly face a "big data" problem. [5]

"Big Data" is about the growing challenge in how we deal with the large and fast-growing sources of data or information. It presents a complex range of analysis and use problems. [6] There are many considerations when dealing with massive amounts of data. One challenge is in having a computing infrastructure that can ingest, validate, and analyze high volumes (size and/or rate) of data. Another is in assessing mixed data (structured and unstructured) from multiple sources. It is often very challenging to deal with unpredictable content with no apparent schema or structure, and often a challenge enabling real-time or near-real-time collection, analysis, and answers. [6]

Before one attempts to extract useful knowledge from data, it is important to understand the steps in the data mining process. Simply knowing many algorithms used for data analysis is not sufficient for successful data mining (DM). The figure below outlines the process of mining data that leads to knowledge discovery.

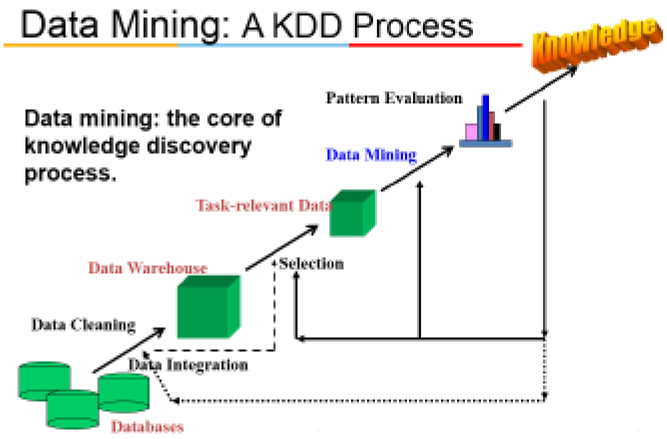


Figure 1: The Knowledge Discovery from Data process allows for the "mining" of valuable knowledge from vast amounts of data just as a miner mines for gold [2]

Fayyad et al. [38] describe the knowledge discovery from data model as a series of nine steps.

1. Develop and understand the application domain. This step includes learning the relevant prior knowledge and considers the goals of the end user.
2. Create a target data set. Here the data miner selects a subset of variables (attributes or features) and data points (examples) that will be used to perform discovery tasks. This step usually includes querying the existing data to select the desired subset.

3. Data cleaning and preprocessing. This step consists of considering outliers, dealing with noise and missing values in the data, and accounting for time sequence information and known changes. Outliers may be irrelevant or be significantly relevant depending on the task at hand.
4. Data reduction and projection. This step consists of finding useful attributes by applying dimension reduction and transformation methods, and finding invariant representation of the data.
5. Choosing the data mining task. Here the data miner matches the goals defined in Step 1 with a particular DM method, such as classification, regression, clustering, etc.
6. Choosing the data mining algorithm. The data miner selects methods to search for patterns in the data and decides which models and parameters of the methods used may be appropriate.
7. Data mining. This step generates patterns in a particular representational form, such as classification rules, decision trees, regression models, trends, etc. More advanced machine learning methods also may apply here.
8. Interpreting mined patterns. Here the analyst performs visualization of the extracted patterns and models, and visualization of the data based on the extracted models.
9. Consolidating discovered knowledge. The final step consists of incorporating the discovered knowledge into the performance system, and documenting and reporting it to the interested parties. This step may also include checking and resolving potential conflicts with previously believed knowledge. In the cyber domain, metrics to measure the effectiveness of detection or battle damage assessment is considered.

The traditional approach to understanding and protecting the cyber domain is a highly manual and human intensive process. It is growing increasingly difficult for these manual processes to keep up with both the massive amount of data and the quickly changing landscape of the cyber domain. It has become necessary to utilize automated techniques to maintain situational awareness and effective offensive and defensive strategies in the cyber realm. Data mining within cyber operations provides some techniques to address these challenges. Through the data mining process described above, one can find hidden patterns, interesting data, or relevant correlations within large datasets. It provides techniques to automate the discovery of structure or patterns which would otherwise be out of reach from human analysts. This analysis is typically performed in an automated process with a variable amount of human interaction, depending on the application.

The scope of data mining for cyber operations is large enough to be its own book, so for purposes of this chapter the scope will be limited to intrusion and malware detection, social networking for cyber situational awareness, and emerging topics for data mining in cyber operations.

Data Mining for Intrusion Detection

Intrusion Detection and Prevention Systems (IDPS) are automated software designed to monitor traffic or mine through select data sources in search of evidence of an intruder attempting to compromise the network. An IDPS is created to monitor characteristics of a host, the network, and combination of both host/network. [9] IDPSs use three basic types of detection to discover intrusions: signature-based detection, anomaly-based detection, and stateful protocol analysis [10].

Signature-based IDPS use signatures, patterns known to indicate a threat, to compare to observable event patterns in order to identify a current threat [10]. A signature-based IDPS is used in firewalls as a first line of defense as it can efficiently identify threats and act before damage is done for very precisely defined and common threats. A disadvantage to this approach is that it relies entirely on a database of known attack signatures to compare against the current network activity. Data mining may be applied to a signature-based IDPS by observing and analyzing known and suspected attacks to discover new signatures and patterns indicative of an intrusion [11].

Applying data mining techniques allows not only for these previously undiscovered signatures to be found, but also for generalized patterns of attacks to be seen. New and novel attacks, which may not exactly match a previously observed signature, may still match the general patterns of an attack that were learned through data mining techniques.

Anomaly-based detection depends on understanding normal patterns of network activity and looking for activity which appears abnormal relative to normal activity [10].The vast majority of new threats will come in as anomalous traffic and yet will likely be undetectable by Signature-based Detectors until new signature rules can be created once they are detected, countered and accounted for in the signature database. An anomaly-based IDPS can be successful in detecting attacks which are novel or vary too far from a signature to be detectable by the signature-based IDPS. They are slow to train and heavily dependent upon having very good "normal" data to upon which to base the training. Data mining is very applicable to this approach, as anomaly detection relies entirely on defining a baseline of normalcy. Various data mining techniques may be effectively used to learn a meaningful definition of normalcy based on known benign network connections. [12]

Stateful Protocol Analysis also looks at behavior outside of known signature patterns to precisely how protocols are designed to be used and what the protocol creators expect to see when those protocols are used [10]. The key is not only in finding anomalous behavior, but also in finding an anomalous behavior beyond what is typical for a specific network activity. Part of understanding a stateful interaction between a user and a network resource is the series of communications between them and not just individual packets as signature-based and most anomaly-based detectors are usually looking at. Looking at the state of the transactions, the intent of the user is revealed. Monitoring state in a network is complex and requires a lot of processing power in high volume networks. As new normal uses for protocols are developed, these systems need to be modified to understand them to ensure that they are not producing false positives. Again, data mining proves useful for defining what constitutes normal use based on previous network activity.

Data Pre-processing

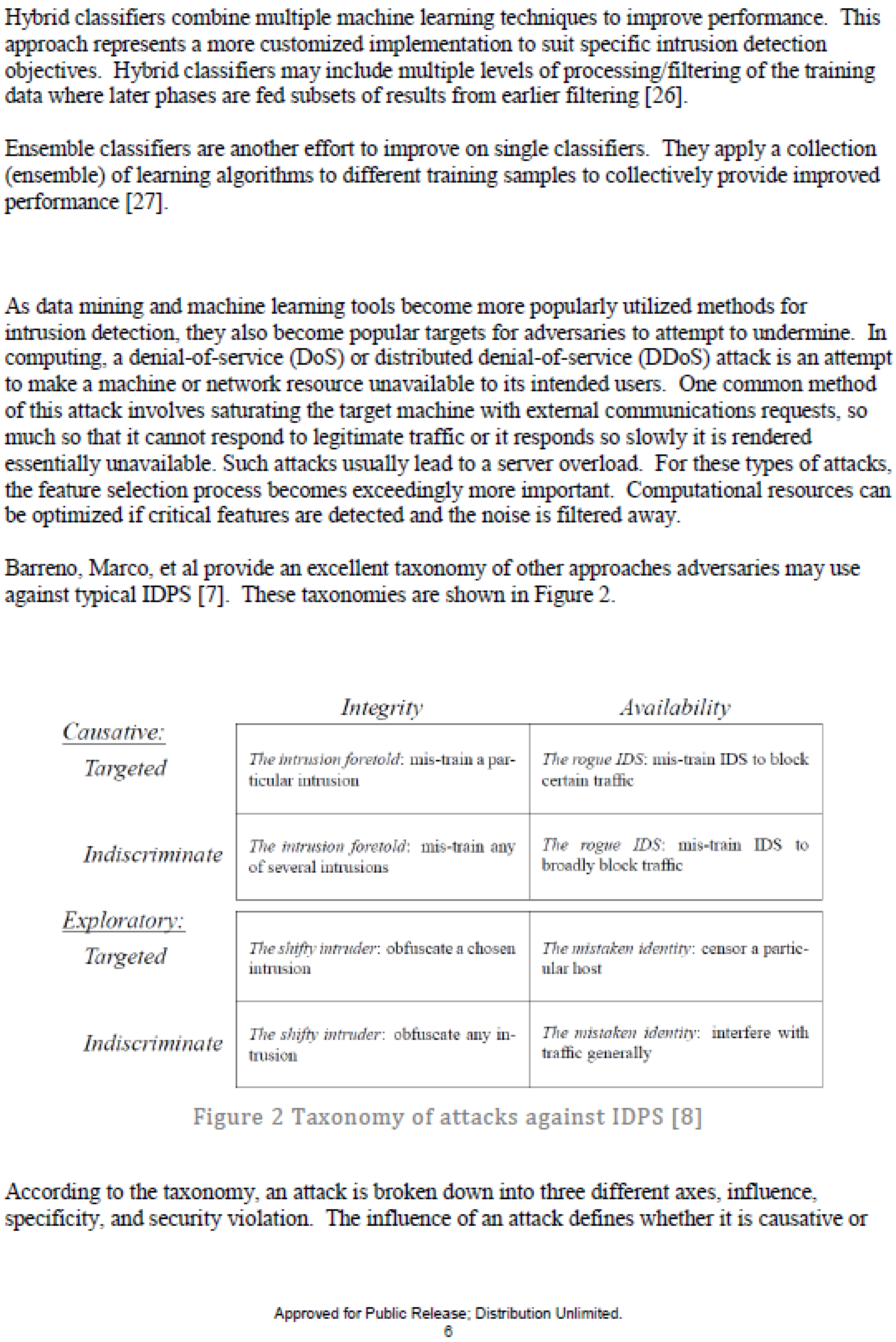
Feature selection is a fundamental part of the Data Mining Process. The main goal is to identify features that are important to the mining effort. The effort of feature selection is to reduce the dimensionality of the data to make processing the data more efficient. Within the study of data mining there is a phenomenon called "the curse of dimensionality" in which all the dataset members appear isolated and unique from the others. According to Dartigue, Jang and Zeng, the areas to analyze for feature selection and extraction can be in [12]:

- Intrinsic features which exist in all network traffic such as protocol, port, destination server name, and requester IP address
- Time-based features which connect traffic from "same host" or "same service" which is valuable in identifying DoS and fast probing exploratory attacks
- Host-based traffic features include grouping connections based upon the same server destination to help to identify slower probing attacks
- Content-based features that are designed to consider long term asynchronous conversations between the target server/service and the attacker's software client. These can be characterized as being slow, methodical and thorough attacks over wide windows in time

Model Development

Various data mining techniques have been explored in existing research to create Intrusion Detection Systems. Tsai, Tsu et al. performed a survey of machine learning techniques for intrusion detection seen in research papers between 2000 and 2007 [13]. Much of the research utilized training data to create classifiers which map input data to an output (benign or an intrusion). New incoming network traffic would be put through this classifier to determine if it represents an intrusion or not. The classifiers were generally one of three types: single, hybrid, or ensemble. Single classifiers utilize one single machine learning algorithm to create a single model which is used to make classifications. The most common single classifiers used to create IDPSs in the research are as follows:

- K-nearest neighbor (KNN) [17][18]: instance based learning to classify a new vector based upon it's calculated nearest neighbor from the training set
- Support vector machines (SVM) [19]: a supervised model defining the decision boundary, gap between the most divergent training examples, based upon support vectors rather than the whole training set to classify new events
- Artificial neural networks (ANN) [20]: information processing units intended to mimic the network of neurons in the human brain for performing pattern recognition
- Self-organizing maps (SOM) [21]: an artificial neural network that uses unsupervised training to produce discretized representation of the training data in the form of a low-dimensional map
- Decision trees [18][22]: maps feature observations about an event to conclusions learned from the features of a training dataset in the form of a classification/regression tree
- Naïve Bayes network [23]: analyzing the features independently of each other along a normal distribution as established by the training dataset
- Genetic algorithms [24]: a meta-heuristic designed to mimic natural selection in finding the most effective classification of new events based upon the features trained from the training dataset
- Fuzzy logic [25]: based upon a real world concept that things are never just black and white, rather they are in the spectrum of grey between the two extremes. It treats the training data as more benign and compares new data to be processed as more or less benign in comparison to the training set.



Hybrid classifiers combine multiple machine learning techniques to improve performance. This approach represents a more customized implementation to suit specific intrusion detection objectives. Hybrid classifiers may include multiple levels of processing/filtering of the training data where later phases are fed subsets of results from earlier filtering [26].

Ensemble classifiers are another effort to improve on single classifiers. They apply a collection (ensemble) of learning algorithms to different training samples to collectively provide improved performance [27].

As data mining and machine learning tools become more popularly utilized methods for intrusion detection, they also become popular targets for adversaries to attempt to undermine. In computing, a denial-of-service (DoS) or distributed denial-of-service (DDoS) attack is an attempt to make a machine or network resource unavailable to its intended users. One common method of this attack involves saturating the target machine with external communications requests, so much so that it cannot respond to legitimate traffic or it responds so slowly it is rendered essentially unavailable. Such attacks usually lead to a server overload. For these types of attacks, the feature selection process becomes exceedingly more important. Computational resources can be optimized if critical features are detected and the noise is filtered away.

Barreno, Marco, et al provide an excellent taxonomy of other approaches adversaries may use against typical IDPS [7]. These taxonomies are shown in Figure 2.

		Integrity	*Availability*
Causative:	*Targeted*	*The intrusion foretold*: mis-train a particular intrusion	*The rogue IDS*: mis-train IDS to block certain traffic
	Indiscriminate	*The intrusion foretold*: mis-train any of several intrusions	*The rogue IDS*: mis-train IDS to broadly block traffic
Exploratory:	*Targeted*	*The shifty intruder*: obfuscate a chosen intrusion	*The mistaken identity*: censor a particular host
	Indiscriminate	*The shifty intruder*: obfuscate any intrusion	*The mistaken identity*: interfere with traffic generally

Figure 2 Taxonomy of attacks against IDPS [8]

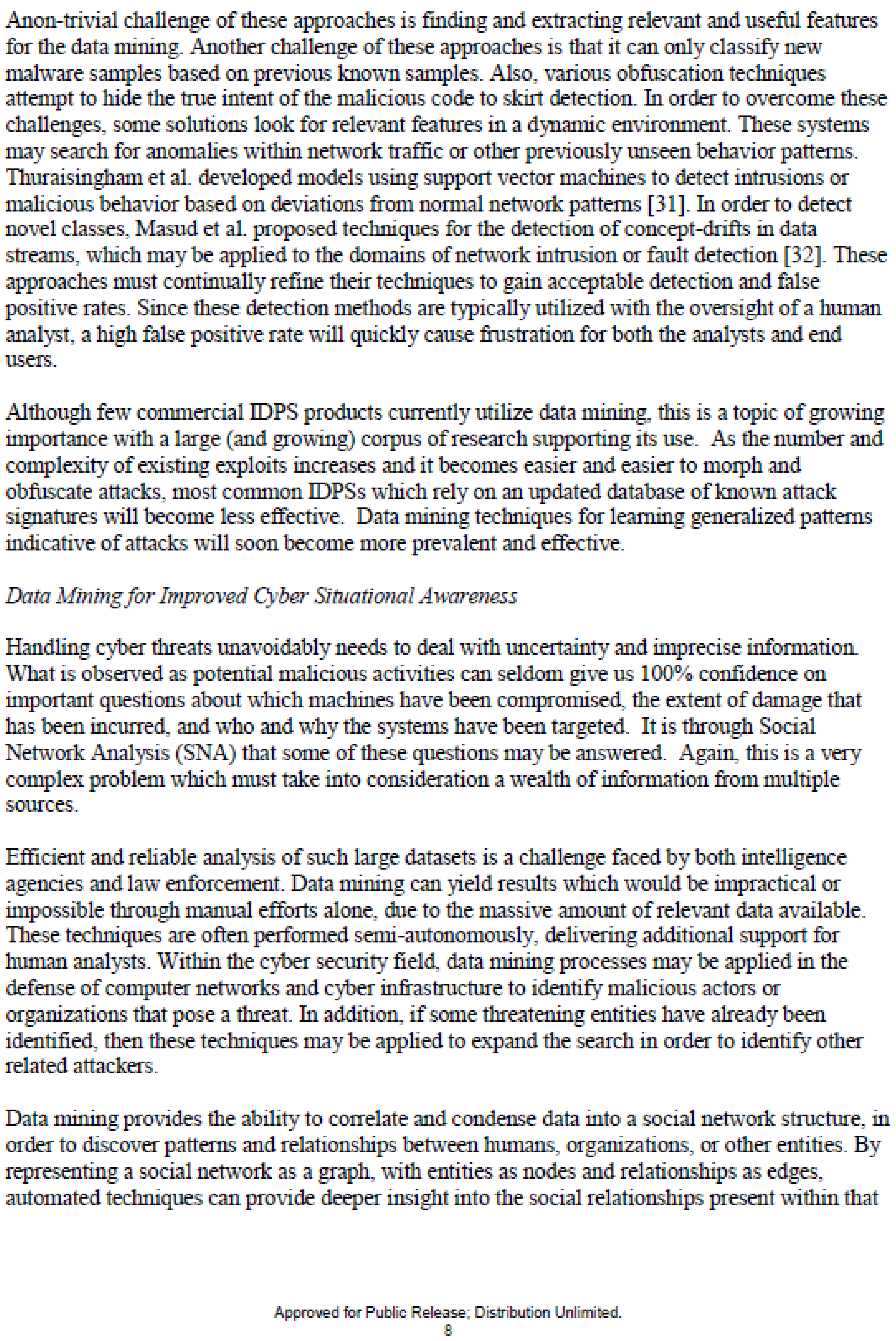
According to the taxonomy, an attack is broken down into three different axes, influence, specificity, and security violation. The influence of an attack defines whether it is causative or exploratory. A causative attack modifies the training set that patterns are mined from in order to influence the learning model. An exploratory attack does not alter the training process, but rather uses other techniques to take advantage of existing weaknesses or blind-spots in the model. An attack is further classified by its specificity as being either targeted or indiscriminate. A targeted attack focuses on a specific intrusion or creating a specific misclassification while an indiscriminate attack looks for any possible intrusion. The third axis, security violation, focuses on the CIA (confidentiality, integrity, availability) model of a network by describing an attack as either an integrity attack or availability attack. An integrity attack results in the IDPS incorrectly classifying an intrusion as benign (false negatives) while an availability attack causes so many misclassifications (both false negatives and false positives) that the IDPS becomes unusable.

Malicious Code Detection

Within the scope of intrusion detection is the more specific security concern of malware or malicious code detection. As the prevalence of malware infections has reached epidemic proportions, it is becoming increasingly important to choose the right defenses to prevent costly malware infections that are targeted at stealing sensitive corporate secrets and mining critical user information records. With today's Internet, malware researchers are seeing a large spike in malware activity and estimate that thousands of new malware variants are being released into the wild daily. Working with large datasets and feature sets to discover hidden patterns has proved extremely applicable to the area of malware detection. Malware can be defined as a program that performs malicious behavior, compromises the security of the system, or performs a function against the wishes of the user. The spread of malware represents an increasing threat to maintaining the security of cyber systems. According to the Symantec Global Internet Security Threat Report, there were over 5,000 reported vulnerabilities in 2012[28].

As mentioned in the previous section, traditional signature based detection is a standard approach for finding and detecting malicious behavior on a system. However, these methods are inherently less effective for detecting novel and polymorphic malware. Signature based detection cannot reliably detect new malware until after it has been identified and given a signature. Polymorphic malware attempts to continuously modify itself in order to evade detection from a previously assigned signature. These concepts pose a serious challenge to existing anti-virus solutions.

Automatic detection of malicious code is a common application of data mining techniques. One method for this detection is through the mining of auspicious binary executables. In order to perform this analysis, appropriate features must be selected to determine whether the sample is benign or malicious. These features may include a list of function calls, strings, headers, byte sequences, or other attributes of the binary [29]. These features can then be processed and fed into a classification algorithm. Some methods assign each sample a classification probability based on the Naive Bayes algorithm, a rules based classifier, or a multi-classifier system [29]. Oulette et al. proposed deep learning algorithms to classify related malware families using a more comprehensive understanding of the malware's intrinsic properties [30].Others have developed solutions which extract n-gram features from both binary and assembly code [31].



Anon-trivial challenge of these approaches is finding and extracting relevant and useful features for the data mining. Another challenge of these approaches is that it can only classify new malware samples based on previous known samples. Also, various obfuscation techniques attempt to hide the true intent of the malicious code to skirt detection. In order to overcome these challenges, some solutions look for relevant features in a dynamic environment. These systems may search for anomalies within network traffic or other previously unseen behavior patterns. Thuraisingham et al. developed models using support vector machines to detect intrusions or malicious behavior based on deviations from normal network patterns [31]. In order to detect novel classes, Masud et al. proposed techniques for the detection of concept-drifts in data streams, which may be applied to the domains of network intrusion or fault detection [32]. These approaches must continually refine their techniques to gain acceptable detection and false positive rates. Since these detection methods are typically utilized with the oversight of a human analyst, a high false positive rate will quickly cause frustration for both the analysts and end users.

Although few commercial IDPS products currently utilize data mining, this is a topic of growing importance with a large (and growing) corpus of research supporting its use. As the number and complexity of existing exploits increases and it becomes easier and easier to morph and obfuscate attacks, most common IDPSs which rely on an updated database of known attack signatures will become less effective. Data mining techniques for learning generalized patterns indicative of attacks will soon become more prevalent and effective.

Data Mining for Improved Cyber Situational Awareness

Handling cyber threats unavoidably needs to deal with uncertainty and imprecise information. What is observed as potential malicious activities can seldom give us 100% confidence on important questions about which machines have been compromised, the extent of damage that has been incurred, and who and why the systems have been targeted. It is through Social Network Analysis (SNA) that some of these questions may be answered. Again, this is a very complex problem which must take into consideration a wealth of information from multiple sources.

Efficient and reliable analysis of such large datasets is a challenge faced by both intelligence agencies and law enforcement. Data mining can yield results which would be impractical or impossible through manual efforts alone, due to the massive amount of relevant data available. These techniques are often performed semi-autonomously, delivering additional support for human analysts. Within the cyber security field, data mining processes may be applied in the defense of computer networks and cyber infrastructure to identify malicious actors or organizations that pose a threat. In addition, if some threatening entities have already been identified, then these techniques may be applied to expand the search in order to identify other related attackers.

Data mining provides the ability to correlate and condense data into a social network structure, in order to discover patterns and relationships between humans, organizations, or other entities. By representing a social network as a graph, with entities as nodes and relationships as edges, automated techniques can provide deeper insight into the social relationships present within that system. SNA techniques help the human analyst discover interesting factors or patterns that have previously unrecognizable. SNA provides mathematical constructs to model and predict useful patterns of social interactions. This analysis can greatly bolster the efforts of human analysts by identifying areas of interest, spotting emerging leaders, and predicting behavior. Krebs utilized SNA to identify core members of a terrorist network involved in the 9/11 attacks [906]. In this example, the relationships and structure were built from surveillance data released by government authorities and publicly available information on the web. This analysis discovered strong mutual connections between the hijackers, while also revealing an emerging leader within the network structure [33].

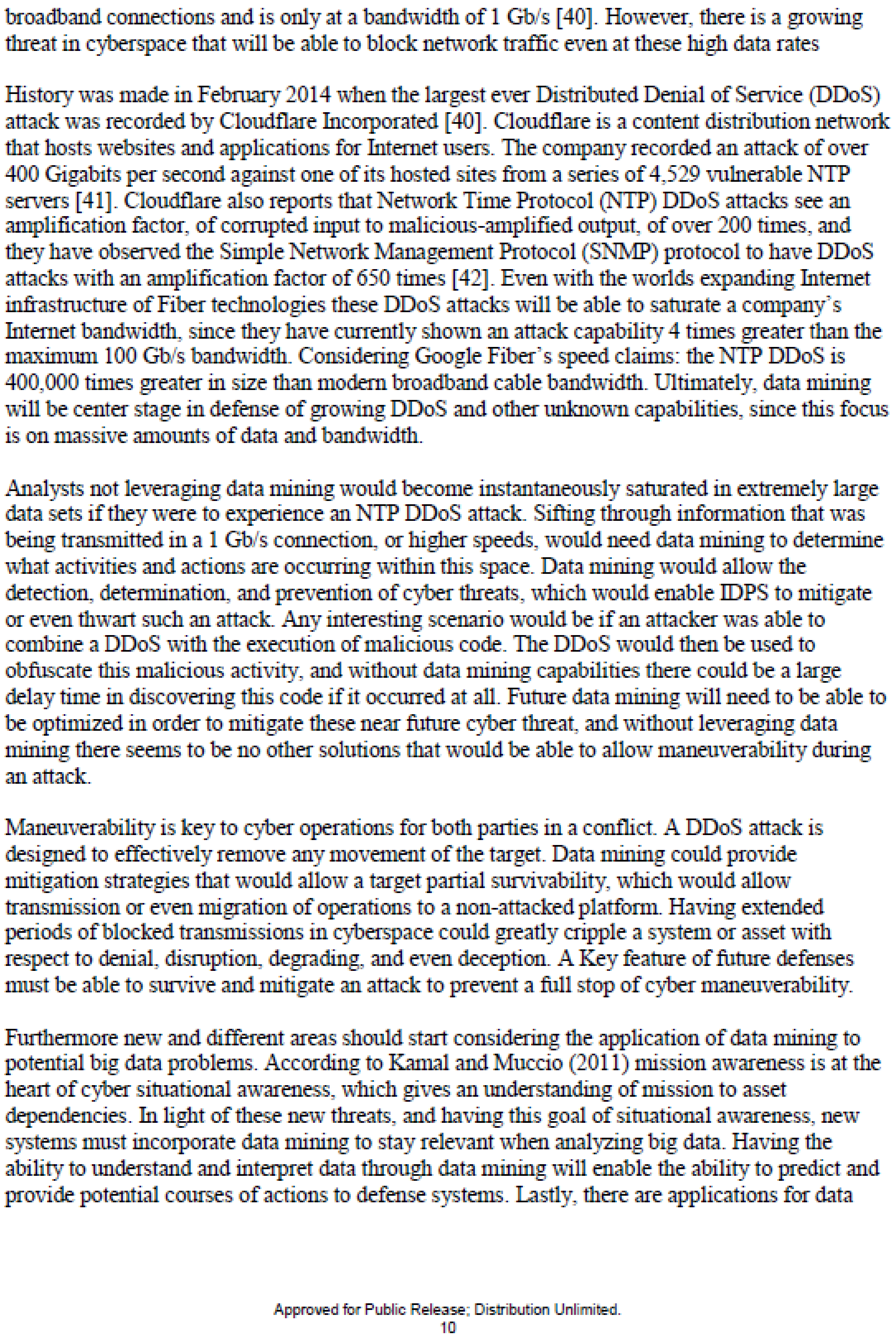
In addition to discovering individual entities within a social network, analysis can reveal the strength and influence of a network as a whole. Shang et al. developed an indicator model that measures the degree of connectivity of a network in order to find and predict criminal networks [34]. Iqbal et al. demonstrated the feasibility of the collecting online chat logs, identifying topics of conversation, and analyzing these messages for possible criminal activity [35]. Chen et al. developed techniques to identify strong subgroups within a network, and to find central members within a subgroup of a potential criminal network [36]. These data mining processes can provide key information in developing a clear understanding of the social dynamics in play within the social network.

In addition to passively understanding the social connections, this analysis can also provide direction for actively influencing the social network. This intelligence may help determine a course of action produce a desired effect within the organization. For example, if the key members of an organization can be identified, then crucial lines of communication may be intercepted or denied to alter the effectiveness of the group. Other techniques may be applied to relevant areas of the graph to achieve a certain desired effect.

This social network analysis often relies heavily on the mining of large datasets to construct these networks. Public social media sites are a common source for this data. Lau et al. produced mining methods which discovered both implicit and explicit relationships derived solely from public social media sites, through extracted words supplied to a probabilistic model [37]. While this analysis can be extremely powerful, it depends strongly on the quality of the data collected. If the data is biased, misrepresented, or incorrect, the results will similarly be erroneous.

Emerging Challenges for Data Mining in Cyber Operations

Modern and emerging networks are rated by the amount of billions of bits they can transport in a second, which uses the metric prefix of giga- to represent a one billion multiplication factor. A common rating of network bandwidth is the term Gigabit, and this rate is abbreviated as Gbps or Gb/s. In a single minute there can up to 60 Gigabits transferred, which is equivalent to 7.5 Giga Bytes and is close to 1.5 DVDs worth of content. This number seems impressive at first, but quickly becomes shadowed when considering there are 1,440 minutes in a day, and the ratification of the IEEE standard 803.3ba defines both a 40 Gb/s and 100Gb/s network [39]. In a single day, at a maximum sustained bandwidth of 100 Gb/s, over 219,142 DVDs worth of content could be transferred. These Internet bandwidth speeds are slowly moving to replace commercial infrastructure as the status quo. Google Fiber advertises it is 100 times the speed of



broadband connections and is only at a bandwidth of 1 Gb/s [40]. However, there is a growing threat in cyberspace that will be able to block network traffic even at these high data rates

History was made in February 2014 when the largest ever Distributed Denial of Service (DDoS) attack was recorded by Cloudflare Incorporated [40]. Cloudflare is a content distribution network that hosts websites and applications for Internet users. The company recorded an attack of over 400 Gigabits per second against one of its hosted sites from a series of 4,529 vulnerable NTP servers [41]. Cloudflare also reports that Network Time Protocol (NTP) DDoS attacks see an amplification factor, of corrupted input to malicious-amplified output, of over 200 times, and they have observed the Simple Network Management Protocol (SNMP) protocol to have DDoS attacks with an amplification factor of 650 times [42]. Even with the worlds expanding Internet infrastructure of Fiber technologies these DDoS attacks will be able to saturate a company's Internet bandwidth, since they have currently shown an attack capability 4 times greater than the maximum 100 Gb/s bandwidth. Considering Google Fiber's speed claims: the NTP DDoS is 400,000 times greater in size than modern broadband cable bandwidth. Ultimately, data mining will be center stage in defense of growing DDoS and other unknown capabilities, since this focus is on massive amounts of data and bandwidth.

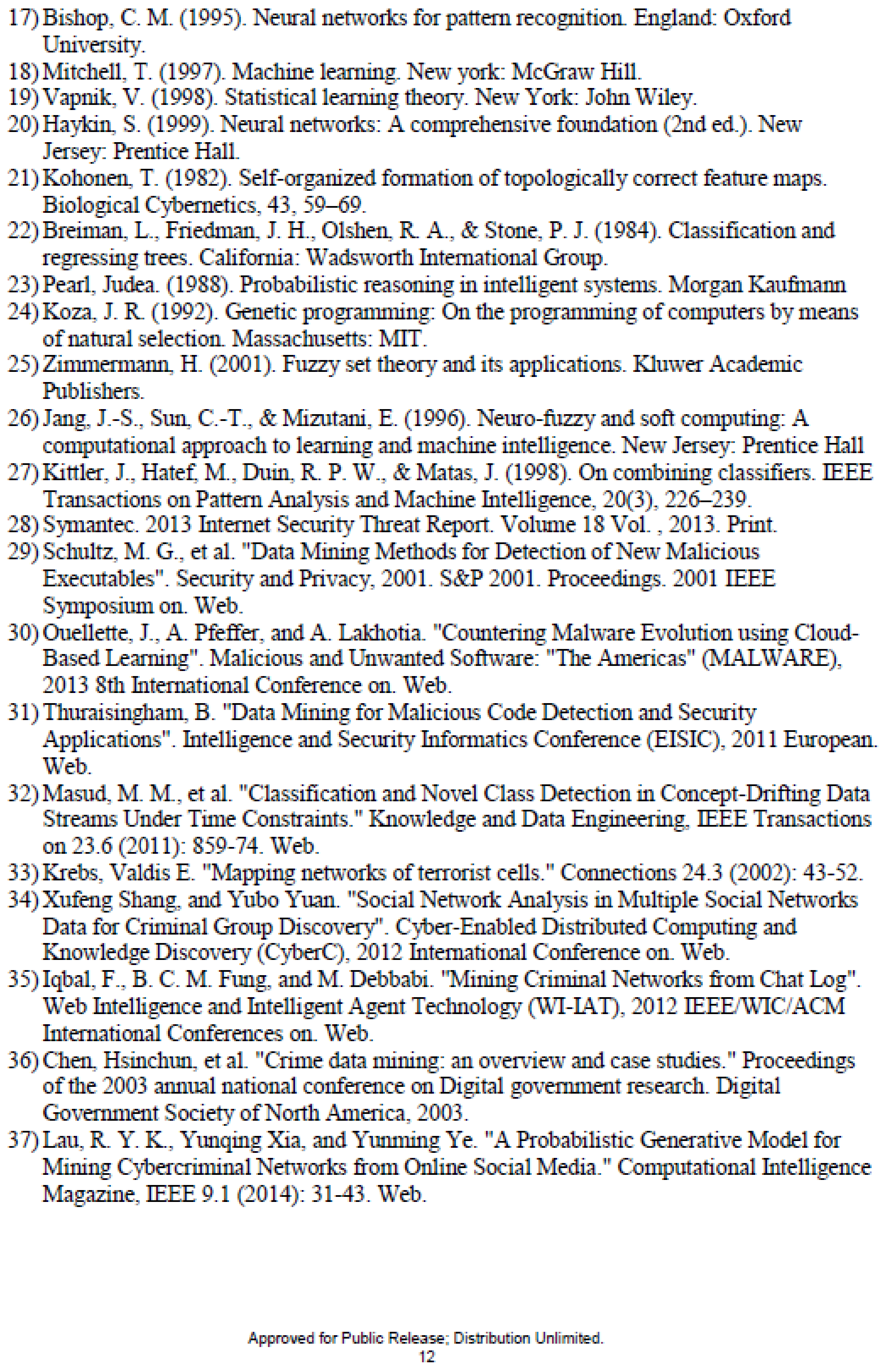
Analysts not leveraging data mining would become instantaneously saturated in extremely large data sets if they were to experience an NTP DDoS attack. Sifting through information that was being transmitted in a 1 Gb/s connection, or higher speeds, would need data mining to determine what activities and actions are occurring within this space. Data mining would allow the detection, determination, and prevention of cyber threats, which would enable IDPS to mitigate or even thwart such an attack. Any interesting scenario would be if an attacker was able to combine a DDoS with the execution of malicious code. The DDoS would then be used to obfuscate this malicious activity, and without data mining capabilities there could be a large delay time in discovering this code if it occurred at all. Future data mining will need to be able to be optimized in order to mitigate these near future cyber threat, and without leveraging data mining there seems to be no other solutions that would be able to allow maneuverability during an attack.

Maneuverability is key to cyber operations for both parties in a conflict. A DDoS attack is designed to effectively remove any movement of the target. Data mining could provide mitigation strategies that would allow a target partial survivability, which would allow transmission or even migration of operations to a non-attacked platform. Having extended periods of blocked transmissions in cyberspace could greatly cripple a system or asset with respect to denial, disruption, degrading, and even deception. A Key feature of future defenses must be able to survive and mitigate an attack to prevent a full stop of cyber maneuverability.

Furthermore new and different areas should start considering the application of data mining to potential big data problems. According to Kamal and Muccio (2011) mission awareness is at the heart of cyber situational awareness, which gives an understanding of mission to asset dependencies. In light of these new threats, and having this goal of situational awareness, new systems must incorporate data mining to stay relevant when analyzing big data. Having the ability to understand and interpret data through data mining will enable the ability to predict and provide potential courses of actions to defense systems. Lastly, there are applications for data mining in current and future cyber modeling, simulation, and war gaming. From participating in these events it has been observed that many Department of Defense war games rely heavily upon analyst input and interpretation of data. The addition of data mining in war games could provide a deeper analysis of the results, or even add the potential of multiple iterations of scenarios where currently there are only a few iterations. The application of data mining to cyberspace is endless, but it provides a greatly exciting future to all of those involved.

Bibliography

1) Jabbour, Kamal, and Sarah Muccio. "The Science of Mission Assurance." Journal of Strategic Security 4.2 (2011).
2) Han, Jiawei, Micheline Kamber, and Jian Pei. Data mining: concepts and techniques. Morgan kaufmann, 2006.
3) Applegate, Scott D. "The principle of maneuver in cyber operations." Cyber Conflict (CYCON), 2012 4th International Conference on. IEEE, 2012.
4) Gregorio-de Souza, Ian, et al. "Detection of complex cyber attacks." Defense and Security Symposium. International Society for Optics and Photonics, 2006.
5) Grant, Tim, Ivan Burke, and Renier van Heerden. "Comparing Models of Offensive Cyber Operations." Proceedings of the 7th International Conference on Information Warfare and Security: Iciw 2012. Academic Conferences Limited, 2012.
6) Villars, Richard L., Carl W. Olofson, and Matthew Eastwood. "Big data: What it is and why you should care." White Paper, IDC (2011).
7) Barreno, Marco, et al. "Can machine learning be secure?." Proceedings of the 2006 ACM Symposium on Information, computer and communications security. ACM, 2006.
8) Barreno, Marco, et al. "The security of machine learning." Machine Learning 81.2 (2010): 121-148.
9) Sabahi, F., and A. Movaghar. "Intrusion detection: A survey." Systems and Networks Communications, 2008. ICSNC'08. 3rd International Conference on. IEEE, 2008.
10) Scarfone, Karen, and Peter Mell. "Guide to intrusion detection and prevention systems (idps)." NIST Special Publication 800.2007 (2007): 94.
11) Han, Hong, Xin-Liang Lu, and Li-Yong Ren. "Using data mining to discover signatures in network-based intrusion detection." Machine Learning and Cybernetics, 2002. Proceedings. 2002 International Conference on. Vol. 1. IEEE, 2002.
12) Lee, W., & Stolfo, S. J. (1998). Data Mining Approaches for Intrusion Detection. Proceedings of the 7th USENIX Security Symposium. San Antonio.
13) Tsai, Chih-Fong, et al. "Intrusion detection by machine learning: A review." Expert Systems with Applications 36.10 (2009): 11994-12000.
14) Dartigue, Christine, Hyun Ik Jang, and Wenjun Zeng. "A new data-mining based approach for network intrusion detection." Communication Networks and Services Research Conference, 2009. CNSR'09. Seventh Annual. IEEE, 2009.
15) Michalski, Ryszard S., Ivan Bratko, and Avan Bratko. Machine Learning and Data Mining; Methods and Applications. John Wiley & Sons, Inc., 1998.
16) Theodoridis, S., & Koutroumbas, K. (2006). Pattern recognition. Amsterdam, Boston, Heidelberg, London, New York, Oxford, Paris, San Diego, San Francisco, Singapore, Sydney, Tokyo: Academic Press



17) Bishop, C. M. (1995). Neural networks for pattern recognition. England: Oxford University.

18) Mitchell, T. (1997). Machine learning. New york: McGraw Hill.

19) Vapnik, V. (1998). Statistical learning theory. New York: John Wiley.

20) Haykin, S. (1999). Neural networks: A comprehensive foundation (2nd ed.). New Jersey: Prentice Hall.

21) Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. Biological Cybernetics, 43, 59–69.

22) Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, P. J. (1984). Classification and regressing trees. California: Wadsworth International Group.

23) Pearl, Judea. (1988). Probabilistic reasoning in intelligent systems. Morgan Kaufmann

24) Koza, J. R. (1992). Genetic programming: On the programming of computers by means of natural selection. Massachusetts: MIT.

25) Zimmermann, H. (2001). Fuzzy set theory and its applications. Kluwer Academic Publishers.

26) Jang, J.-S., Sun, C.-T., & Mizutani, E. (1996). Neuro-fuzzy and soft computing: A computational approach to learning and machine intelligence. New Jersey: Prentice Hall

27) Kittler, J., Hatef, M., Duin, R. P. W., & Matas, J. (1998). On combining classifiers. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(3), 226–239.

28) Symantec. 2013 Internet Security Threat Report. Volume 18 Vol. , 2013. Print.

29) Schultz, M. G., et al. "Data Mining Methods for Detection of New Malicious Executables". Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on. Web.

30) Ouellette, J., A. Pfeffer, and A. Lakhotia. "Countering Malware Evolution using Cloud-Based Learning". Malicious and Unwanted Software: "The Americas" (MALWARE), 2013 8th International Conference on. Web.

31) Thuraisingham, B. "Data Mining for Malicious Code Detection and Security Applications". Intelligence and Security Informatics Conference (EISIC), 2011 European. Web.

32) Masud, M. M., et al. "Classification and Novel Class Detection in Concept-Drifting Data Streams Under Time Constraints." Knowledge and Data Engineering, IEEE Transactions on 23.6 (2011): 859-74. Web.

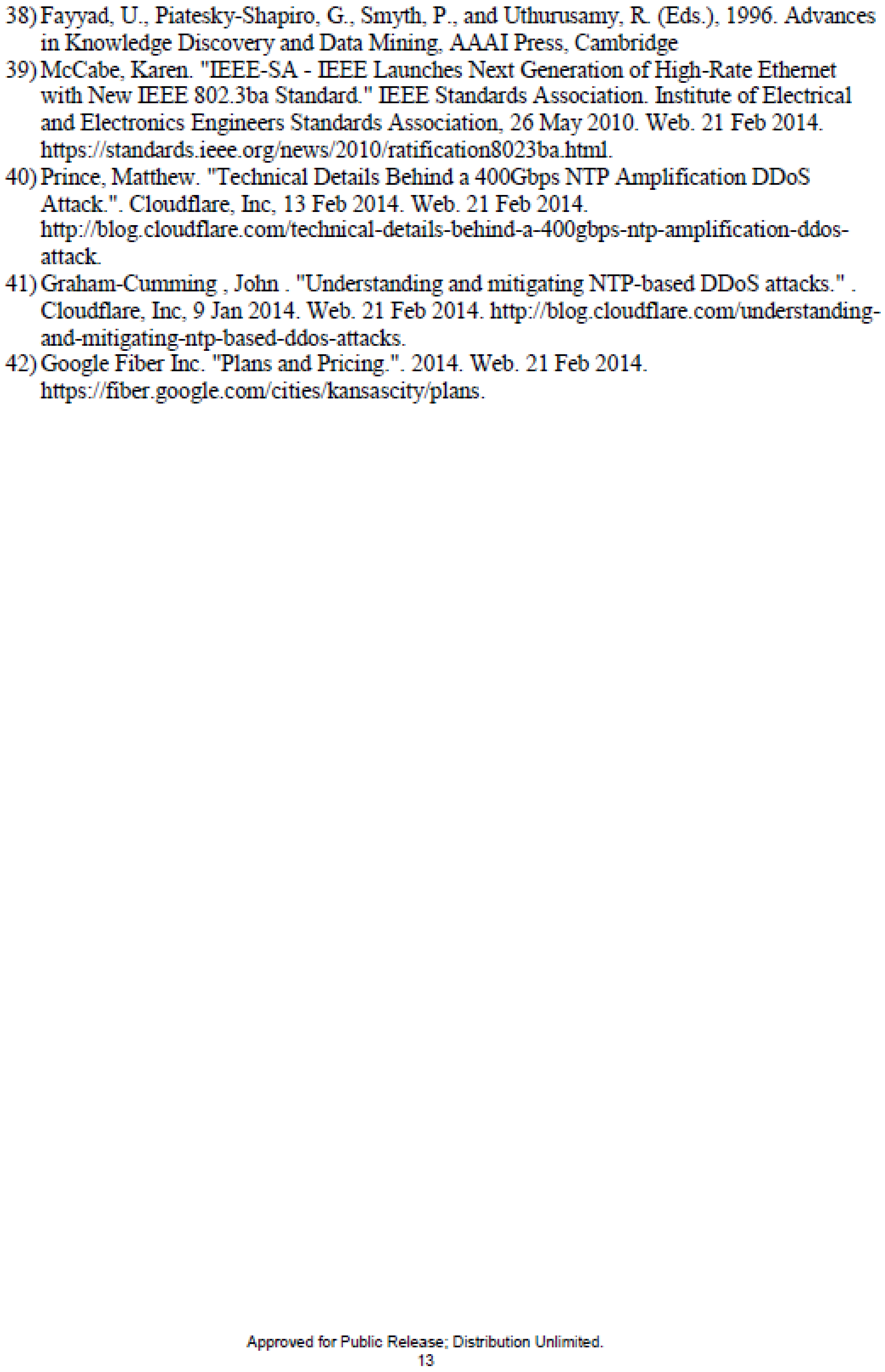
33) Krebs, Valdis E. "Mapping networks of terrorist cells." Connections 24.3 (2002): 43-52.

34) Xufeng Shang, and Yubo Yuan. "Social Network Analysis in Multiple Social Networks Data for Criminal Group Discovery". Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2012 International Conference on. Web.

35) Iqbal, F., B. C. M. Fung, and M. Debbabi. "Mining Criminal Networks from Chat Log". Web Intelligence and Intelligent Agent Technology (WI-IAT), 2012 IEEE/WIC/ACM International Conferences on. Web.

36) Chen, Hsinchun, et al. "Crime data mining: an overview and case studies." Proceedings of the 2003 annual national conference on Digital government research. Digital Government Society of North America, 2003.

37) Lau, R. Y. K., Yunqing Xia, and Yunming Ye. "A Probabilistic Generative Model for Mining Cybercriminal Networks from Online Social Media." Computational Intelligence Magazine, IEEE 9.1 (2014): 31-43. Web.



38) Fayyad, U., Piatesky-Shapiro, G., Smyth, P., and Uthurusamy, R. (Eds.), 1996. Advances in Knowledge Discovery and Data Mining, AAAI Press, Cambridge

39) McCabe, Karen. "IEEE-SA - IEEE Launches Next Generation of High-Rate Ethernet with New IEEE 802.3ba Standard." IEEE Standards Association. Institute of Electrical and Electronics Engineers Standards Association, 26 May 2010. Web. 21 Feb 2014. https://standards.ieee.org/news/2010/ratification8023ba.html.

40) Prince, Matthew. "Technical Details Behind a 400Gbps NTP Amplification DDoS Attack.". Cloudflare, Inc, 13 Feb 2014. Web. 21 Feb 2014. http://blog.cloudflare.com/technical-details-behind-a-400gbps-ntp-amplification-ddos-attack.

41) Graham-Cumming , John . "Understanding and mitigating NTP-based DDoS attacks." . Cloudflare, Inc, 9 Jan 2014. Web. 21 Feb 2014. http://blog.cloudflare.com/understanding-and-mitigating-ntp-based-ddos-attacks.

42) Google Fiber Inc. "Plans and Pricing.". 2014. Web. 21 Feb 2014. https://fiber.google.com/cities/kansascity/plans.

APPENDIX B - Evaluating data distribution and drift vulnerabilities of machine learning algorithms in secure and adversarial environments (SPIE DSS 2014)

Evaluating data distribution and drift vulnerabilities of machine learning algorithms in secure and adversarial environments

Kevin Nelson[a], George Corbin[a], Dr. Misty Blowers[b]
[a]BAE Systems, 581 Phoenix Drive, Rome, NY, USA 13441; [b]Air Force Research Laboratory, 525 Brooks Rd., Rome, NY USA 13441

Abstract

Machine learning is continuing to gain popularity due to its ability to solve problems that are difficult to model using conventional computer programming logic. Much of the current and past work has focused on algorithm development, data processing, and optimization. Lately, a subset of research has emerged which explores issues related to security. This research is gaining traction as systems employing these methods are being applied to both secure and adversarial environments. One of machine learning's biggest benefits, its data-driven versus logic-driven approach, is also a weakness if the data on which the models rely are corrupted. Adversaries could maliciously influence systems which address drift and data distribution changes using re-training and online learning. Our work is focused on exploring the resilience of various machine learning algorithms to these data-driven attacks. In this paper, we present our initial findings using Monte Carlo simulations, and statistical analysis, to explore the maximal achievable shift to a classification model, as well as the required amount of control over the data.

Keywords: Adversarial Machine Learning, Intrusion Detection, Monte Carlo, Hidden Markov Models

Introduction

The primary job of a security administrator is to secure resources at every level from unauthorized access and intrusions. It is critical to any organization to secure the network against adversaries both external and internal. In the course of implementing defensive measures for network protection against malicious adversaries, an administrator needs to understand the approach adversaries take in trying to compromise their systems. This knowledge goes a long way towards enabling the administrator to harden the network against unauthorized access and intrusions. Increasingly, Machine Learning-based Intrusion Detection Systems (ML-IDS) are used in this capacity[0]. An ML-IDS uses machine learning (ML) techniques to analyze network traffic to create a model which specifies either general patterns of normalcy or those indicative of an intrusion. It is important to have a greater understanding of the strengths and weaknesses of common ML algorithms in order to optimize defensive capabilities of ML-IDS. This paper discusses the risks involved with utilizing ML in an adversarial environment, such as intrusion detection, and presents an evaluation of a toy example, an anomalous color detector which utilizes Hidden Markov Models (HMMs).

Background

Machine learning and data/concept drift

Machine learning is a branch of artificial intelligence which focuses on allowing a computer to learn representative patterns and rules from sample training data or past experiences which may be generalized to solve specific problems. An ML system becomes better at its job over time by iteratively performing its operations.

Often, an ML system is burdened with large non-stationary streams of data that drift over time[0]. This is also referred to as “concept drift.”[0,0,0,0] This is a concern as it means that new data is being provided that may inadvertently be misclassified by an outdated model. The solution is to include re-training for the ML system into the other ongoing processes in place to maintain classification and prediction accuracy. This will ensure that user interactions with changing functions are included in the models used by the ML system. When ML is used in applications which utilize continuous cycles of retraining and model updating from new data input they are referred to as online ML.

Machine learning in an adversarial environment

Unfortunately, ML systems sometimes operate in environments that include an adversary. Such environments commonly listed in other research include Spam filtering[0], Intrusion Detection Systems[0] and fraud detection systems[0]. These are environments in which an opponent would gain an advantage by finding a way to avoid or otherwise subvert the established ML system. Adversarial learning takes place when an ML system undergoes training with data collected from an adversarial environment[0]. In nearly every case, there is no known or obvious opponent that is providing adversarial data to an online ML system’s training data. In an unsupervised ML system, there is the possibility of such data effectively poisoning the training dataset and thus introducing a weakness that must be mitigated[0]. Often, as mentioned above, ML systems utilize retraining and online learning to account for general concept drift and to avoid a high rate of false positives, which further expands the risk of an adversary tainting the training data. This is a factor that few ML researchers take into account while developing new algorithms and techniques. Recently however, this topic has begun to gain traction and the field of adversarial ML has begun to emerge. Specifically Huang et al define adversarial ML as “the study of effective machine learning techniques against an adversarial opponent.”[0]

Intrusion detection systems

Of the several adversarial environments where we find ML used, the IDS is the most exciting and undergoing the greatest change in research. The two ways a network and its resources can be secured are to: a) preventatively implement rules/policies for use and apply patches to applications to avoid obvious and readily fixed flaws in security; b) provide a reactive system to intercept security violations as they are occurring. An IDS is one such reactive measure. An IDS is an application, and in some cases an appliance, that is designed to monitor network and/or computer system operations. The IDS is designed to monitor for malicious activity or other usage policy violations which may present a security threat to the network or computer system being monitored and is given rules for how to respond to these violations based upon severity[0]. The technique used by an IDS to detect malicious activity falls into one of two categories: signature-based detection or anomaly-based detection. Signature-based detection, also known as misuse detection, examines network traffic and log files, searching for specific patterns, or signatures, that are known to be indicative of an intrusion[0]. An anomaly-based IDS utilizes a baseline of normal network behavior, which it either learns or has specified by an administrator. This baseline is made up of heuristics or rules that describe what constitutes normal behavior[0]. Intrusions are then identified by detecting a statistical divergence from the norm, rather than looking for specific signatures.

Machine learning-based intrusion detection

An IDS that uses ML to handle creation of rules and processing of data according to those rules is called an ML-IDS. Ariu identifies protecting web applications with IDS as necessary and a "tricky task" as "they are in general large, complex and highly customized." Ariu further states that signature based IDS are not able to defend adequately in the face of zero-day attacks and the complex rules that would be required for complex web applications. As he suggests, anomaly-based solutions have the greatest potential and these are found in ML-IDS[0]. Barreno et al. agree with this assessment as they see the strength of an ML-IDS is in being able to identify "novel differences in traffic."[0] ML techniques have proven to be effective for anomaly-based intrusion detection, as they provide a method for automatically learning a model of normalcy that new incoming traffic can be evaluated against[0]. As the field of ML continues to explode in popularity, and as anomaly-based IDSs continue to become more in demand due to their ability to detect never before seen attacks, ML-IDSs will soon become much more prevalent[0].

ML-IDS in an adversarial environment

Unfortunately, IDSs tend to operate in an adversarial environment in which an opponent would reap a great benefit by cleverly causing misclassifications, an aspect which most ML researchers do not take into account. The greatest concern, as suggested above, is that data drift will be introduced to the training data for an ML-IDS which uses unsupervised learning. This could result in creating rules which overfit the training data leading to false positives due to concept drift or, worse yet, new training data at the edges of the detection domain producing new models which allow previously flagged malicious communication to be allowed by the IDS[0]. This last concern is the greatest threat to a network's defense and is the threat the simulations in this paper are concerned with understanding better. There are essentially three sources we have identified that may cause data drift:

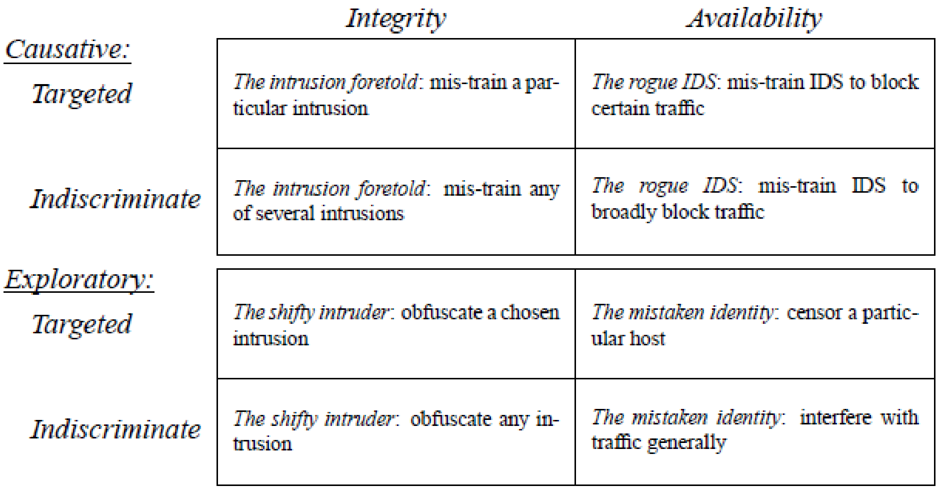
1. Random noise and concept drift from the network and users
2. Adversarial drift due to nonspecific malicious probing exploring the network which is building what a penetration tester calls a "footprint" of the network[0,0]
3. Adversarial drift due to specific targeting of the network by an adversary in a persistent and threatening manner trying to measure the efficacy of rules and possibly even to insert edge data to influence training to later allow malicious connections as authentic[0]

The goal of an ML-IDS is to provide accurate protection which is robust and uses generalized models. In order to do this, it must accurately differentiate between normal data drift and noise, leaving the resources defended uncorrupt and uncompromised. Thus, due to the increased prevalence of anomaly-based ML-IDSs operating in adversarial environments, it is our belief that it is essential to analyze the underlying anomaly detection ML algorithms utilized by these IDSs to determine their resilience to adversarial drift.

Adversarial ML taxonomy

Barreno et al. created a taxonomy to categorize potential attacks an adversary may employ against an ML system. Table 1 below is reproduced from the paper "The security of machine learning."[0]

Table 1: Taxonomy of attacks against ML systems with examples (captured from Barreno et al.[0])



	Integrity	*Availability*
Causative:		
Targeted	*The intrusion foretold*: mis-train a particular intrusion	*The rogue IDS*: mis-train IDS to block certain traffic
Indiscriminate	*The intrusion foretold*: mis-train any of several intrusions	*The rogue IDS*: mis-train IDS to broadly block traffic
Exploratory:		
Targeted	*The shifty intruder*: obfuscate a chosen intrusion	*The mistaken identity*: censor a particular host
Indiscriminate	*The shifty intruder*: obfuscate any intrusion	*The mistaken identity*: interfere with traffic generally

This taxonomy was designed to address how an attack can affect the ML system. An attack has three dimensions: the Influence, the Security Violation and the Specificity of the attack. Influence is either causative or exploratory; the Security Violation attacks either integrity or availability; and the Specificity is either targeted or indiscriminate[0].

Integrity of the system is a measure that demonstrates the authenticity of data as having not been altered from origination or otherwise corrupted by malicious or accidental means. An Integrity attack is destructive in nature. Availability is a measure that represents how readily the data is accessed and used as intended by authorized users for intended and authorized purposes. An availability attack is a denial of service, either specific or large-scale[0].

The approach adversaries use may be causative, in which they take actions to bring about changes in the learning model through influence over the training data, or else exploratory in which their actions simply probe/investigate for potential weaknesses that can be exploited with another action. These causative and exploratory actions can be of two forms: targeted and indiscriminate. Targeted and indiscriminate attacks differentiate in their specificity and scope.

A targeted causative attack against integrity is an attack which chooses a specific intrusion and attempts to alter the ML training to make the ML system model allow this specific intrusion. This type of attack has been addressed very little in existing research. Therefore, we believe that it is essential to study the risk of an ML algorithm to such an attack. The remainder of this paper details initial experiments to analyze the risk to an HMM algorithm, the results of which may be built upon for future experiments and analysis.

Experiment

Goals

Our work focused on evaluating an anomaly-based ML algorithm's resilience to targeted causative integrity attacks. Under this type of attack, an adversary attempts to cause a specific point which is otherwise classified as anomalous by the learning model to be misclassified as normal through influence over the training data. The initial experiment focused on evaluating the extent to which an adversary could affect the algorithm's model and the amount of control an adversary would require over the training data in order to cause misclassifications in the resulting learning model.

Assumptions

In a true adversarial environment, the amount of information available to an adversary is often limited. It is reasonable to assume that the attacker will not know the exact data set used for training the classification model. However, an attacker may very well be aware of general trends in the data and what generally represents “normal” data. For example, in the field of spam detection, “clean” words tend to stay somewhat constant from organization to organization and common spam databases are freely available online to give an idea of what is generally flagged. From this, a clever adversary may be able to create a plan for tricking a spam filter. Therefore, in our experiments, we assume that our “normal” data used to train the classifier comes from a probability distribution which the adversary is aware of. Our initial experiments utilize training data consisting of numerical values randomly generated from a Gaussian (normal) distribution.

Additionally, our experiments were run under the contamination assumption[0] – the adversary is able to send points that the algorithm will use during training. This assumption was implemented in two different ways. In the first scenario, the adversary is able to arbitrarily add custom points to the training set as desired. This may be the case if the attacker somehow has access to the database and wishes to discreetly add points, or if the adversary has knowledge of when initial data collection and training will be taking place, before a classification model has been implemented, and inserts specific training points at that time. This presents a worst case scenario for the algorithm to defend against. In the second scenario, the algorithm is an online learner which continually retrains with the new data presented to it. However, it only accepts data for retraining which is classified as normal by the previously existing model. Therefore, any insertion points must appear to be benign, presenting a greater obstacle for an adversary.

Monte Carlo simulations

Since our experiments rely on randomly generated data from a distribution, Monte Carlo experiments are used to obtain results. Monte Carlo experiments find solutions to problems by accumulating and aggregating the results of multiple runs using random data from the same probability distribution, also known as repeated random sampling. Due to time and equipment constraints, the number of iterations used in our experiments was limited.

Hidden Markov Models

Our initial experiments focused on the evaluation of Hidden Markov Models (HMMs), an ML algorithm commonly used for anomaly detection. An HMM consists of hidden states which follow the Markov property and have associated initial and transition probabilities. In addition, an HMM consists of observed variables, with each variable having a certain probability of occurring in each hidden state. Sequences of observed variables are used to train the HMM and learn the hidden state and observed variable probability matrices. Once learned, these probabilities are used to determine the probability score of new test observation sequences[0]. Again, due to time and equipment limitations, other ML algorithms are not evaluated in this paper. However a general approach has been created as a result of this work, which will greatly simplify future suggested evaluations.

Experiment setup

In order to illustrate the above concepts and concerns, we conducted experiments utilizing an anomaly detector designed to classify color values as either normal or anomalous based on their RGB values. The RGB value of a color is the extent of red, green, and blue the color contains, represented as integer values between 0 and 255 inclusive. Therefore, each training point is a vector of integers of length three. Each training point's red, green, and blue values were sampled from a Gaussian distribution with a mean of 127 and a standard deviation of 30. Figure 1 displays 2000 colors randomly sampled from this distribution, plotted on the Cartesian axes. The X axis represents the red value, the Y axis represents the green value, and the Z axis represents the blue value.

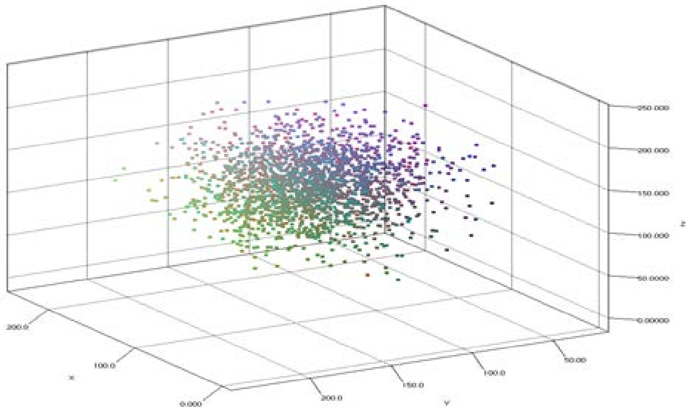


Figure 1: 2000 random colors from a Gaussian distribution

The classifier is then created by using color values sampled from this distribution to train an HMM. Future points are tested against the resulting HMM and given a probability score. If the score is below a certain threshold, then the point is classified as anomalous. Repeatedly training HMMs with points from this distribution, and then testing the probability scores of points from the same distribution against the resulting model, consistently revealed 9.0x10^{-9} to be an appropriate threshold value which would cause points close to the mean to be classified as normal and those far from the mean to be classified as anomalous. This threshold value was used for all subsequent experiments.

The JAHMM library[0] was used in our experiments to train and test the HMM models. We chose to use three hidden states and used a K-Means clustering based learner to initialize the probability matrices. The Baum-Welch algorithm is then run for ten iterations to fully train the HMM and the forward algorithm is used to determine the probability of new points.

Two separate targeted causative integrity attacks were investigated. The first involved an adversary who desires for the color green-yellow (RGB value of [173, 255, 47]) to be classified as normal, and in the second, the adversary desires goldenrod (RGB value of [255, 193, 37]) to be classified as normal.

Results and discussion

Initially, experiments were carried out to determine the probability score of the adversary's colors when tested against an HMM trained with a varying number of points sampled from the Gaussian distribution. For each training set size, 100 iterations of sampling, training, and testing were run. The mean and median values of those iterations are presented in Figure 2. These experiments revealed that the test score of the points is not dependent on the number of points in the training set. Therefore, the same threshold value may be held constant for all ensuing experiments with this distribution, regardless of the number of points used. The average probability score of green-yellow over all experiments was 1.60x10^{-11} and the average probability score of goldenrod was 1.75x10^{-12}, each of which is well below the set threshold value of 9.0x10^{-9}. Also of note, is that the median score for both colors remained at a value of zero until there were roughly 24,000 training points. This is due, in part, to the mechanics of an HMM. If any of the test point's values were not seen during training, then the test is much more likely to return a probability of 0 for the entire point. For example, if a value of 255 is not encountered during training, then both green-yellow and goldenrod will likely have a probability of 0. As the number of training points increases, the likelihood of encountering each value in the test colors at least once increases.

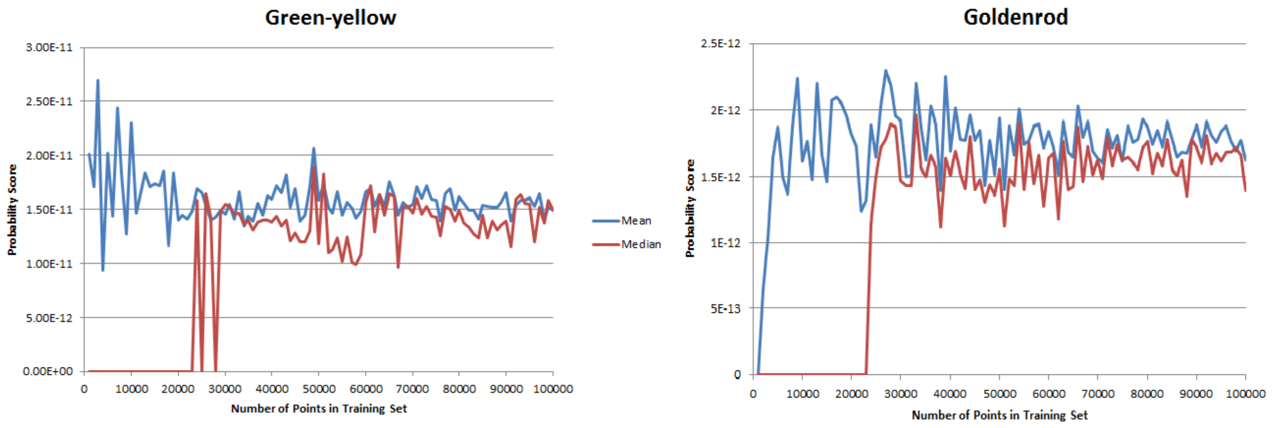


Figure 2: Initial probabilities of selected anomalous points

Next, experiments were conducted to determine the effect an adversary could have through the insertion of a single point, in terms of causing the chosen outliers to appear more probabilistically normal. Our first implementation of the contamination assumption was used during this experiment, meaning that the adversary was free to add any arbitrary point to the training data set. The point which would have the greatest impact on the HMM's ability to detect the specific anomaly was determined by solving the following equation:

$$x^* = \underset{x}{argmax}\, g(x_0)_x - g(x_0) \qquad (1)$$

where *x* represents a potential insertion point, x_0 represents the attacker's chosen anomaly, $g(x_0)$ represents the probability of x_0 according to the HMM trained with its current training set, and $g(x_0)_x$ represents the probability of x_0 according to the HMM trained with point *x* injected into the training set.

A particle swarm optimization algorithm[0] was used to quickly find a rough solution to the equation. PSO works by placing a certain number of "particles" into the solution space which then, for a certain number of iterations, travel to new locations searching for an optimal solution. Particle movement is dictated by mathematical calculations based on the particle's position and velocity, the best point seen by the particle, and the best point seen by any particle. The PSO was implemented using the JSwarm library, and was configured to use 20 particles and run for 20 iterations.

The training set consisted of a varying number of color values sampled from our Gaussian distribution in order to determine the correlation between training set size and the adversary's ability to alter an anomalous point's probability with a single insertion point. For each training set, the HMM was trained and the anomalous point's probability score was determined. Then, an insertion point was determined by solving equation 1 using PSO, the point was added to the training set, the HMM was retrained, the anomalous point's new probability score was computed, and the difference between the two probability scores was calculated. For each training set size, 10 iterations were run, and the mean values of those iterations are displayed in Figure. The results reveal a function nearly identical to the one displayed in the following equation:

$$eff = c * n^{-1} \tag{2}$$

where *eff* is the maximum effect of a single point, *c* is a constant, and *n* is the number of points in the training set.

The effect that an adversary can have on the probability score of an anomalous point using a single insertion point has an almost perfect inverse relationship with the number of points in the training set. This negative correlation implies that increasing the size of the training set may mitigate some of the risk of a targeted causative attack against an ML algorithm. However, the time and resource cost of training an ML algorithm increases as the number of training points increases, and at some point, this cost must be weighed against the potential impact of an adversary.

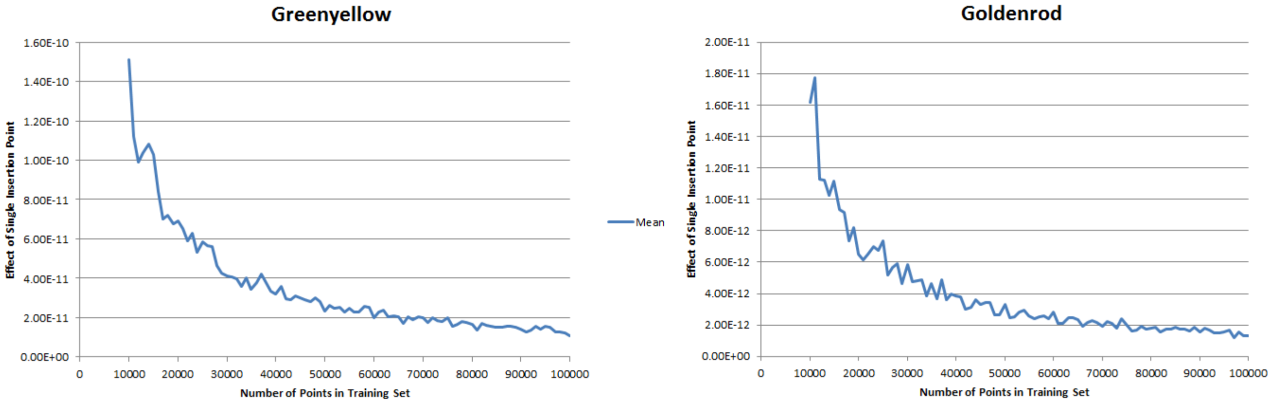


Figure 3: Effect of a single insertion point on selected anomalous points

Experiments were then run to determine exactly what effort would be required by an adversary in order to trick the HMM into believing that the chosen anomalous colors were, in fact, normal. The goal of the experiment was to learn how susceptible an HMM is to a targeted causative integrity attack. In the first set of trials, the adversary is again given the ability to arbitrarily add points to the training set. It is assumed that points can only be added to the data set, not removed, and that the adversary is the only one adding data. For varying size training sets, the process described in Figure 4 is followed in order to determine the number of points an adversary would have to inject in order to cause the desired misclassification. In the process, *I* is equal to the list of injection points selected by the adversary, $g(x_0)_I$ is equal to the probability of x_0 according to an HMM with *I* injected into the training set, and γ is equal to the probability threshold.

```
$initialize\ I = \emptyset$
$while\ g(x_0)_I < \gamma$
    $x^* = \underset{x}{argmax}\ g(x_0)_{I+x} - g(x_0)_I$
    $add\ x^*\ to\ I$
$return\ |I|$
```

Figure 4: Process to determine effort required by adversary under scenario 1

For each training set size, this iterative process was carried out 10 times, and the results were aggregated. Figure 5displays the mean values of these runs. The relationship between the size of the training set and the number of insertion points required to create a misclassification is relatively linear, which implies that increasing the amount of points in the training set will steadily increase the amount of points necessary for an adversary to force specific misclassifications. However, the percent of control over the training set that the attacker needs remains fairly constant. In order to force green-yellow to be misclassified, the attacker needed to insert 0.3701% of the original training set on average and in order to force goldenrod to be misclassified, the attacker needed to insert 0.7344% of the training set on average. The number of points necessary to insert and the percent of the training set necessary to control appears to be related to the initial probability of the anomalous point. Goldenrod, which had a lower initial probability, required nearly twice as much control over the training set.

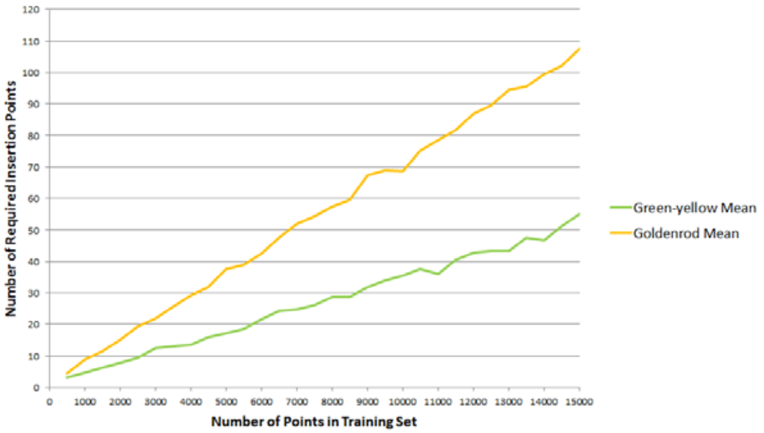


Figure 5: Effort required by adversary to cause misclassification of selected anomalous points under scenario 1

The contamination assumption was then limited somewhat for the next set of experiments. In these trials, it is assumed that a classification model has already been trained and is operational, and that the adversary does not have direct access to the data base. The classifier uses an online learner which continually retrains when presented with new input, but only if the new input is classified as normal by the existing model. This requires the adversary to craft points which appear normal, but are designed to be similar to the selected anomalous point in order to slowly shift the probability models of the learner, causing the anomalous point to appear normal. Under this scenario, the process shown in Figure 6 was followed using various size training sets in order to determine the number of points an adversary must insert to cause a specific misclassification.

```
initialize $I = \emptyset$
while $g(x_0)_I < \gamma$
        $x^* = \underset{x}{argmax}\, g(x_0)_{I+x} - g(x_0)_I \ s.t.\ g(x)_I > \gamma$
        add $x^*$ to $I$
return $|I|$
```

Figure 6: Process to determine effort required by adversary under scenario 2

Due to time and equipment limitations, this process was only carried out for 5 iterations for each training set size. Figure 7 displays, for both target colors, the mean values of these iterations under this scenario, as well as the results of the previous scenario for comparison. Again in this scenario, there appears to be a linear relationship between the number of points in the training set and the number of points an adversary must inject to cause the desired misclassification. However, requiring that new input points be classified as normal before being used for retraining nearly doubles the effort required by an adversary. On average, the adversary required 0.3701% control of the training set in scenario one and 0.6153% control in scenario two in order to force a misclassification of green-yellow. In scenario one, goldenrod required 0.7344% control of the training set, and in scenario two it required 1.0981% control. The technique utilized in scenario two of validating the normalcy of a point before including it in the training set offers a potential strategy for mitigating the risk of a causative attack against an ML-based anomaly detector by increasing the amount of control required by the adversary. This approach, however, may not allow for natural concept drift, unless it is introduced slowly over time, without occurring too drastically different from the norm.

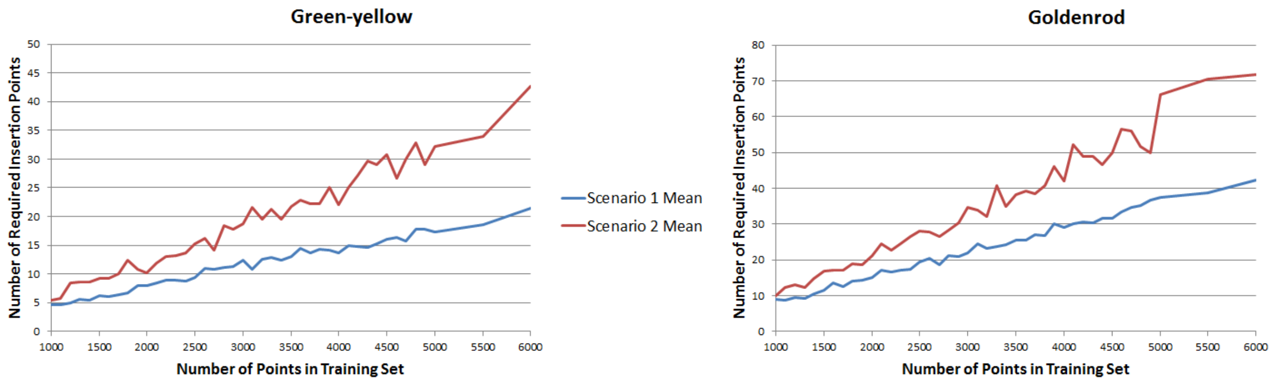


Figure 7: Effort required by adversary to cause misclassification of selected anomalous points under scenario 1 and 2

Also of note in these experiments is that if all values from the anomalous color do not appear at least once during training, then it will be highly unlikely to achieve a probability score above zero. For example, when the adversary's target color is green-yellow, the values 173, 255, and 47 should all be in the training set. If, let's say, 255 did not appear in the training set, then all hidden states in the HMM would likely have an associated probability of zero for that value, which would cause any sequence of values containing 255 to have a probability of zero. The adversary would not be able to change this via insertion, because any points containing 255 would be flagged. Therefore, training sets where the anomalous point had an initial probability of zero were not considered in these experiments, meaning that, on average, the anomalous point had a slightly higher initial probability under scenario two than under scenario one.

Conclusions

In this paper, we discussed an emerging field of research known as adversarial ML. We argue that due to its proven ability to discover and generalize patterns, ML will continue to become more widely used in operational environments. However, these environments often include an adversary, a challenge which is rarely accounted for during the development of the ML algorithms. One such technology that utilizes ML in an adversarial environment is an IDS. It is essential that an IDS remains secure, and we argue that understanding the potential for an adversary to subvert the underlying ML algorithms is an important step towards improving a network's defensive capabilities.

We conducted base-line experiments in order to better understand and demonstrate what effort and level of control over the training data an adversary would require in order to conduct a targeted causative integrity attack against an ML algorithm. These experiments utilized a Hidden Markov Model in order to detect anomalous colors based on their RGB values. Training data was drawn from a Gaussian distribution and Monte Carlo simulations were run in order to provide aggregate results. The experiments reveal an inverse relationship between the number of points contained in the training set and the potential negative impact of a single insertion point. Additionally, there appears to be a linear relationship between the number of points in the training set and the amount of effort required by an adversary to cause a desired misclassification. The trends and conclusions identified from these experimental results are summarized in Table 2.

Table 2: Trends and conclusions

Trend	Explanation
Larger training set improves defensive capabilities	As the training set size increases, this increases the requisite number of points an adversary needs to produce in order to induce adversarial drift. This increases the effort required by the adversary to calculate the necessary points and to introduce them into the actual data set.
Highly anomalous points require more effort by the adversary	The experiments showed that goldenrod, which had an average initial probability of $1.75 \text{x} 10^{-12}$ required nearly twice as many insertion point as green-yellow which had an average initial score of $1.60 \text{x} 10^{-11}$.
Machine learning algorithms not overly secure from adversarial drift	In the experiments above, as the training set size increased, the amount of points required by the adversary increased. However, the required percent control over the training set remained relatively constant. This value was below 1% in the first test cases despite the test points being considerably well below the anomaly threshold.
Requiring insertion points to be classified benign before inserting into training set increases adversary's effort.	In the second set of experiments, we observed that the number of points required by the adversary to cause adversarial drift at least doubles from the number required in the first set.

Future research areas based on this research should include analysis of different ML algorithms to verify the trends identified in this paper and datasets based on alternative data distributions and feature spaces. Additionally, Barreno et al. describe various defensive measures which may be utilized to harden an ML system[0]. These techniques include:

- Reject on negative impact (RONI) defense – measures effects of each training instance and rejects points which are seen to have a negative impact on classification

- Robust algorithms – based upon Robust Statistics, the goal is to create a procedure which will limit the impact of deviant points by accounting for qualitative robustness, the breakdown point and the influence function of the procedure
- Online learning with experts – uses a set of classifiers each designed to provide a different security property and predictions/advice for training
- Hide training data – if access to the training data is denied, an adversary is unable to determine the exact decision boundaries of the models used by the ML so as to analyze a way to bypass them
- Good feature selection – make classifiers difficult to reverse engineer through careful selection of features which are kept secret and possibly even mapping raw features into a different feature space altogether
- Limited/misleading feedback – provide feedback to attacker that provides as little information as possible revealing their level/lack of success during the probing attack.

Future research should take into account and analyze these defensive measures and include an investigation into the efficacy of the defensive measures when applied to specific ML algorithms. Utilizing the analysis method described in the above experiments, future experiments may be run in order to paint a better picture of the defensive capabilities of ML algorithms in adversarial environments.

References

Ariu, D. “Host and Network based Anomaly Detectors for HTTP Attacks.” Diss. PhD thesis, PhD Program in Electronic and Computer Eng. (DRIEI), University of Cagliari, Italy. (2010)

Kantchelian, A., et al. "Approaches to adversarial drift." Proceedings of the 2013 ACM workshop on Artificial intelligence and security. ACM. (2013)

Kantarcioglu, M., Xi, B., and Clifton, C. "A Game Theoretical Framework for Adversarial Learning." CERIAS 9th Annual Information Security Symposium. (2008)

Dalvi, Nilesh, et al. "Adversarial classification." Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM. (2004)

He, H., and Garcia, E. A. "Learning from imbalanced data." Knowledge and Data Engineering, IEEE Transactions on 21.9: 1263-1284. (2009)

Nelson, B., et al. "Exploiting Machine Learning to Subvert Your Spam Filter." LEET 8: 1-9. (2008)

Kloft, M., and Laskov, P. "Online anomaly detection under adversarial impact." (2011)

Barreno, M., et al. "The security of machine learning." Machine Learning 81.2: 121-148. (2010)

Huang, L., et al. "Adversarial machine learning." Proceedings of the 4th ACM workshop on Security and artificial intelligence. ACM. (2011)

Dua, S., and Du, X. Data Mining and Machine Learning in Cybersecurity. CRC press. (2011)

Ahmed, F., Johnson, T., and Tsui, S. "From measurements to metrics: PCA-based indicators of cyber anomaly." SPIE Defense, Security, and Sensing. International Society for Optics and Photonics. (2012)

Corona, I., Tronci, R., and Giacinto, G. "SuStorID: A multiple classifier system for the protection of web services." Pattern Recognition (ICPR), 2012 21st International Conference on. IEEE. (2012)

Chung, S. P., and Mok, A. K. "Advanced allergy attacks: Does a corpus really help?" Recent Advances in Intrusion Detection. Springer Berlin Heidelberg. (2007)

Fogla, P., and Lee, W. "Evading network anomaly detection systems: formal reasoning and practical techniques." Proceedings of the 13th ACM conference on Computer and communications security. ACM. (2006)

Rabiner, L. "A tutorial on hidden Markov models and selected applications in speech recognition." Proceedings of the IEEE 77.2: 257-286. (1989)

François, J.-M. "JAHMM - Hidden Markov Model (HMM): An implementation in Java." URL: http://www.runmontefiore.ulg.ac.be/~francois/software/jahmm (2006)

Kennedy, J. and Eberhart, R. “Particle Swarm Optimization.” IEEE 0-7803-2768-3/95 (1995)

APPENDIX C - Evaluating Model Drift in Machine Learning Algorithms (IEEE CISDA 2015)

Evaluating Model Drift in Machine Learning Algorithms

Kevin Nelson, George Corbin, Mark Anania,
Matthew Kovacs, and Jeremy Tobias
BAE Systems
Rome, NY, USA

Dr. Misty Blowers
Air Force Research Laboratory
Rome, NY, USA

Abstract— **Machine learning is rapidly emerging as a valuable technology thanks to its ability to learn patterns from large data sets and solve problems that are impossible to model using conventional programming logic. As machine learning techniques become more mainstream, they are being applied to a wider range of application domains. These algorithms are now trusted to make critical decisions in secure and adversarial environments such as healthcare, fraud detection, and network security, in which mistakes can be incredibly costly. They are also a critical component to most modern autonomous systems. However, the data driven approach utilized by these machine learning methods can prove to be a weakness if the data on which the models rely are corrupted by either nefarious or accidental means. Models that utilize on-line learning or periodic retraining to learn new patterns and account for data distribution changes are particularly susceptible to corruption through model drift. In modeling this type of scenario, specially crafted data points are added to the training set over time to adversely influence the system, inducing model drift which leads to incorrect classifications. Our work is focused on exploring the resistance of various machine learning algorithms to such an approach. In this paper we present an experimental framework designed to measure the susceptibility of anomaly detection algorithms to model drift. We also exhibit our preliminary results using various machine learning algorithms commonly found in intrusion detection research.**

Keywords—adversarial machine learning; cyber security; intrusion detection systems; model drift

Introduction

A security administrator maintains a constant vigilance over their network resources using a variety of security tools. Most security plans include signature-based detection and prevention systems in order to effectively protect against known threats and mitigate known vulnerabilities. While daily updates and changes to security policy, processes, and functions tend to be minimal and focused upon anticipating and intercepting system compromises, the administrator's single greatest fear is the threat of zero day exploits. Zero day exploits are based upon currently unknown and unpatched vulnerabilities that conventional signature-based protections are not equipped to handle. Machine learning based intrusion detection systems (ML-IDS) are increasingly being implemented to account for the inadequacies of signature-based methods [1][2]. These machine learning (ML) solutions are quickly becoming a popular choice due to ML's ability to generalize, learn representative patterns indicative of intrusions, create representations of normalcy, and detect anomalies straying from the established models. Unfortunately, as these ML-IDS operate and learn patterns in a highly contested environment, they become susceptible to corruption. Either through inadvertent or deliberate means, corrupted data may be introduced into the learning model, resulting in model drift [3][4]. This is a key limitation of ML systems and requires the developer to understand the inherent weaknesses of the underlying learning processes.

In this paper, we present a systematic experimental framework based around Monte Carlo simulations, designed to measure the resilience of machine learning algorithms to model drift. In addition, we discuss the preliminary results of experimental iterations run against several anomaly detection algorithms as well as the effects of model drift on an open-source intrusion detection system which relies on these methods.

Background

Machine Learning

Machine learning is a branch of artificial intelligence which focuses on allowing a computer to learn representative patterns and rules from sample training data or past experiences which may be generalized to solve specific problems. Due to its advanced ability to automatically glean meaningful information from large datasets, ML is quickly appearing more and more in fielded systems. As it becomes more trusted, ML is increasingly being applied within secure environments to make critical decisions and identify threats [4][5][6][7][8].

Intrusion Detection Systems

An intrusion detection system (IDS) is an application or appliance that is designed to monitor network and/or computer system operations for malicious activity and other usage policy violations that present a security threat to the monitored system. An IDS traditionally uses predefined rules based on known threats in order to assess the severity of the threat and react accordingly. These types of systems are known as signature-based IDSs. Alternative methods involving the use of machine learning, however, have recently gained popularity and are expected to continue to do so [4]. This is due to ML's ability to overcome signature-based detection's greatest flaw which is the inability to detect novel or zero-day attacks. The most popular forms of ML-IDS utilize anomaly-detection, a method in which algorithms first build a model of normalcy using benign network traffic and then evaluate new traffic against the model to detect behaviors or patterns deviating from this established normalcy [2][8]. These methods have been shown to be effective at identifying potential threats to the system [2][9]. Anomaly detection is often the popular choice over other machine learning techniques such as binary classification because it is often difficult to attain recent and representative malicious samples on which to train the models.

Adversarial Machine Learning

As stated before, machine learning algorithms are sometimes deployed within adversarial environments such as intrusion detection, spam filtering and fraud detection where they are required to secure a system from unauthorized access [5]. Unfortunately, due to the nature of unsupervised machine learning algorithms, they are heavily targeted by the adversaries they are intended to protect against. This is because such a great advantage can be gained by the adversary if they are able to subvert and infiltrate the system through the system's own ML processes. This is a factor few ML researchers take into account while developing new algorithms and techniques [4].

The authors of [3] have created a three dimensional taxonomy designed to address how an attack can affect an ML based system. These dimensions are Influence, the Security Violation and the Specificity of the attack. The dimensions are broken down further into sub-categories: Influence is either causative or exploratory; the Security Violation targets either integrity or availability; and the Specificity is either targeted or indiscriminate. Our research mainly focuses on studying an ML system's resistance to a *Targeted Causative* attack against the *Integrity* of the learning system. In such an attack, an adversary chooses a specific anomalous point and makes an effort to influence the ML system models to misclassify the test point.

This misclassification by the system is induced by inserting crafted points into the ML algorithm's training data to drift the learning model in a desired direction. This is a form of what is known as adversarial drift. This process creates rules which may overfit the current training data, leading to false positives, or worse yet, creates new models from data at the edge of the detection domain and allows previously flagged malicious data into the system. The latter is of great concern, especially for an ML-IDS.

We have identified three main sources from which model drift, which is any shift in the established baseline of normalcy, can occur:

- Random noise and concept drift from the normal network traffic and its users.
- Adversarial drift from nonspecific, malicious probing of the network to create what penetration testers call a "footprint."
- Adversarial drift from specific targeting of the network in a persistent and threatening manner to measure the efficacy of rules and possibly even insert edge data to influence training to later allow malicious connections as authentic ones.

Unfortunately for system administrators, it is often incredibly difficult to differentiate between normal model drift over time and deliberate adversarial model drift. Additionally, new patterns and model drift are often accounted for through online learning or by periodically retraining the models. This gives an adversary an ideal means for inserting malicious data into the learning models [4].

Experiment Framework

Our work focused on studying the resilience of an IDS's underlying ML algorithms to induced model drift. In this section, we describe an experimental framework that was created as a result of this work for gathering statistical measurements and comparing various algorithms.

Goals

The main goals of our work were to develop a framework to study and test ML algorithms which have or may be used in intrusion detection systems. This research was done in an effort to demonstrate the strengths and weaknesses of these algorithms in regards to their susceptibility to adversarial drift and offer suggestions for why and in what ways they can be effectively used in an IDS. We intended to create a generic, universal framework to allow for the simple incorporation of additional algorithms using various training data sources. We wanted to be able to compare multiple algorithms, multiple implementations of similar or identical algorithms and to create repeatable experiments. Through experimental iterations, the framework should provide statistical measurements and analysis that better inform a system administrator.

Assumptions

In an authentic network, an intruder would have limited access to information about an ML-IDS which has been deployed. While we certainly have a concern for external threats, the worst case scenario is an insider threat or an intruder who has already gained access to the internal network. We base our initial assumptions on this perspective. In order to have a worst case scenario baseline where a potential adversary would have near full control over the IDS and all associated and requisite resources, several assumptions were made to start with.

We operate under what is known as the contamination assumption [10]. This means that the adversary has the ability to insert points that will be used during retraining of the ML-IDS. As mentioned above, model drift is often accounted for through the use of retraining and online learning, so it is not unreasonable to assume an adversary could take advantage of this window to insert data. For our work, we limit this assumption to create a slightly more realistic case in which insertion points must be classified as benign by the existing classifier in order to be included. We assume that the adversary would wish to remain undetected by including only points that are not flagged as suspicious and can be allowed into the re-training dataset via normal traffic through the monitoring ML-IDS. The contamination assumption is necessary if the adversary desires to induce drift in the models and force the ML-IDS to consider a point as normal that previous models would have otherwise considered anomalous and blocked.

In initial experiments, we assume the adversary has full knowledge of the IDS, its classification algorithms, the training data, and the results of classification. From this worst case scenario, we could potentially dial back the assumed knowledge and access privileges an adversary might have in order to gain a more realistic view of the measured weaknesses for each algorithm in an adversarial environment.

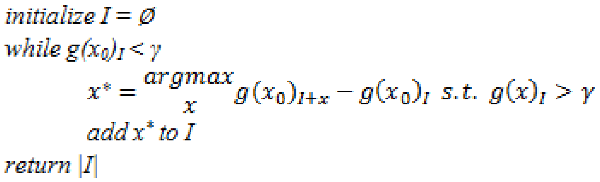
Approach

We chose to design our framework around Monte Carlo simulations, which is an experimental method that relies on repeated random sampling. This helps eliminate data specific results by finding averages across multiple runs, decreasing the variance in our results. To this end, an application program interface (API) was created with interfaces that allow for simply running Monte Carlo experiments and adapting new algorithms and data sources for testing.

To set up an experiment, an ML-IDS was identified and its source code and documentation were analyzed to identify the underlying ML algorithms, features included in monitoring and training, as well as any perceived constraints in operation. Research was performed to determine if prior investigations of the selected algorithms existed, including documented optimization approaches that could be used. After identifying the type of data processed by the ML-IDS, an anomalous point was identified with which to test the algorithm within the simulator. The simulator API was extended as appropriate to handle the new algorithm implementation library, the data type the ML-IDS is designed to process, and the known vulnerability that would be used to model a live threat in the experimentation.

Upon successful integration of the algorithm and data source into the API, the framework allows for experiments to be run to identify precisely what effort an adversary would need to expend in order to force a misclassification on the selected test point through model drift created by the introduction of crafted insertion data into the training set. These experiments may be repeated for varying training set sizes, different percentages of control the adversary has over the retrain data, and varying algorithm-specific parameters values. For each configuration, multiple iterations are run with randomly sampled data from the selected data source and the results are aggregated and placed into graphs for further analysis. This provides the user with an overall picture of the resistance of the algorithm to adversarial drift and may be used to compare algorithms.

The approach selected for the adversary in the experiments represents a worst case for the defender in which each new point introduced by the adversary is added to the training set and causes a retrain, as long as the point appears benign to the existing classifier. The adversary chooses the point, which when added to the training set causes the greatest increase in the probability of normalcy of the target point. The general form of this approach is summarized in Fig. 1 in which I is equal to the list of injection points selected by the adversary, g(x0)I is equal to the probability of x0 according to the model with I injected into the training set, and γ is equal to the anomaly threshold.



```
initialize $I = \emptyset$
while $g(x_0)_I < \gamma$
        $x^* = \underset{x}{argmax}\, g(x_0)_{I+x} - g(x_0)_I \;\; s.t. \;\; g(x)_I > \gamma$
        add $x^*$ to $I$
return $|I|$
```

Fig. 1. Experimental approach to determine effort required by an adversary to force misclassifications

The framework allows us to test both algorithm-specific and generic optimization approaches for selecting the insertion points introduced by the adversary. A generic approach, such as genetic algorithms that repeatedly retrain the algorithm to find the point that has the most optimal effect on the test score of the target point, requires less prior knowledge of the algorithms and can therefore be applied to a large variety of ML algorithms [7][11]. However, the repeated retrains often take a large amount of time, which is infeasible with insufficient resources. Algorithm-specific approaches require in-depth knowledge of the algorithms, but run faster and often create points which have a greater effect. For the experiments described below, due to time and resource constraints, we chose to develop algorithm-specific methods for selecting insertion points.

EXPERIMENT

In order to test the validity of our proposed approach for analyzing the resistance of an IDS's machine learning algorithms to induced model drift, a number of experimental runs were performed. This section describes the various algorithms that were investigated, including an open-source IDS which relies on machine learning, and the experimental procedure.

Centroid Anomaly Detector

The first algorithm studied was a simple centroid anomaly detector [12]. This algorithm is trained by simply finding the empirical mean of the training examples. An unlabeled data point is then tested by calculating its Euclidean distance to the mean, or centroid. If this distance is greater than a determined threshold, then the point is considered to be anomalous. Calculation of a data point's anomaly score is summarized in (1).

$$f(x) = \left\| x - \frac{1}{n}\sum_{i=1}^{n} x_i \right\| \qquad (1)$$

Despite its incredible simplicity, this algorithm is popular in various security applications [12]. We chose to run initial experiments with the centroid anomaly detector due to its low computation time, applicability, and its ability to be easily comprehended and visualized. Its primary purpose is to be used as a baseline experiment to demonstrate our approach.

Additionally, the optimal undetected approach is relatively straightforward from the adversary's perspective. The adversary determines the vector between the target anomalous point and the centroid and inserts a point along this vector at a distance from the centroid equal to the anomaly threshold value. This approach is illustrated in Fig. 2.

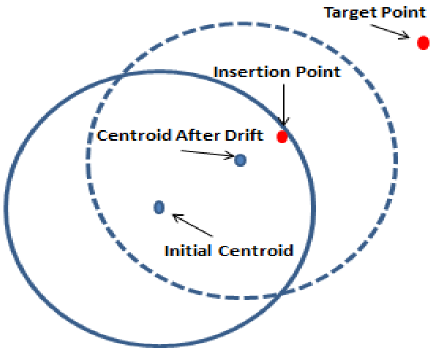


Fig. 2. Illustration of the optimal drift strategy against a centroid anomaly detector

Support Vector Machines

After these baseline experiments, we proceeded to study one-class support vector machines (SVMs). An SVM is a machine learning model which is designed to create a separating plane between two classes of data, and a one-class SVM is a special case which separates normal from anomalous. In order to do this, the one-class SVM first maps the training data into the kernel space, in our case using the Gaussian kernel shown in (2), and then finds the hyperplane which separates a desired fraction of the training points from the origin. This algorithm was chosen due to its relative simplicity and common appearance in ML-IDS research [2][13][14].

$$K(x,y) = \Phi(x)\cdot\Phi(y) = \exp(-\gamma\|x-y\|^2) \qquad (2)$$

During our experimentation with this algorithm, we developed a quick method to determine the insertion point which will have the near-optimal drift effect from the adversary's perspective. The adversary simply finds the support vector nearest to the target anomalous point, and inserts the point along the vector between the two that is closest to the target anomaly without being flagged as anomalous.

Hidden Markov Models / HMMPayl

Next, we focused on the evaluation of Hidden Markov Models (HMMs). An HMM is a machine learning model consisting of hidden states which follow the Markov property and have associated initial and transition probabilities. In addition, an HMM consists of observed variables, with each variable having a certain probability of occurring in each hidden state. Sequences of observed variables are used to train the HMM and learn the hidden state and observed variable probability matrices. Once learned, these probabilities are used to determine the probability score of new test observation sequences. This algorithm was chosen again due to its relative simplicity as well as its common appearance in ML-IDS research [9][15][16].

During this study, we also investigated HMMPayl, an open-source network IDS which utilizes HMMs to detect anomalies. HMMPayl inspects network packet payloads represented as byte strings, using the n-grams from these byte strings to train its models. In an effort to increase classification accuracy, HMMPayl uses an ensemble of HMMs, combining the results from each to make its predictions [9].

Through an investigation of the algorithm, we developed a quick method and heuristic to determine the adversary's near-optimal insertion points. Since the sequence probabilities returned by an HMM are largely driven by the frequency of individual symbols in the training set, we chose to focus our approach on these frequencies. We wanted to target the specific n-grams from the test point that contained symbols common in the point but also had low probabilities, meaning that they likely contained symbols uncommon in the training set. Therefore, to create the adversary's insertion points, for a set number of iterations we iteratively added symbols to the insertion point according to (3) where *T(x)* is the number of times symbol x occurs in the test point and *Tr(x)* is the number of times x appears in the training set. Then, if the constructed insertion point is considered anomalous by the existing model, symbols in the point are iteratively replaced by the symbol occurring most frequently in the training set until the point is no longer flagged.

$$x^* = \underset{x}{\arg\max} \frac{T(x)}{Tr(x)^2} \tag{3}$$

Procedure

Two sets of experiments were run using the Monte Carlo simulation framework. In the first set, the above described algorithms were used to classify color values as either normal or anomalous based on their RGB values. The RGB value of a color is the extent of red, green, and blue the color contains, represented as integer values between 0 and 255 inclusive. Therefore each training point is a vector of integers of length three. The training sets used for these experiments were sampled from a Gaussian distribution with a mean of 127 and a standard deviation of 30. We then selected colors that the selected classifiers identified as anomalous, and measured the effort required by an adversary to force misclassifications using the outlined approach. While this data set is overly simplistic, it allows us to easily illustrate the above described concepts and to identify general trends.

The second set of experiments was designed to be more realistic, testing the resiliency of the actual IDS to adversarial drift using network data. For these experiment runs, we utilized the DARPA'99 dataset [17] to train the anomaly detection models and used the HTTP attack dataset from [18] as our test points on which to force misclassifications. Although the DARPA'99 dataset is outdated and has been widely criticized [19], it is still highly appropriate and valuable for our needs. It is the most common public dataset used to baseline ML-IDS and lends to the repeatability of our experiments. Additionally, the largest complaint against DARPA'99 is that it is no longer suitable for measuring the accuracy of an IDS and its ability to detect intrusions. However, this is not the context for which we are using it. Our goal was to find appropriately formatted network data and some attack point that is flagged as anomalous by the classifier trained on this data. This approach does not consider the overall accuracy of the classifier, but rather its ability to continue making an accurate prediction despite adversarial attempts to induce model drift.

Results and Discussion

In this section we discuss the preliminary results of the above described experiments, highlight the statistical measurements our framework allows the user to gather, and identify general trends.

For the first set of experiments using the color RGB data, we selected two colors that were found to be anomalous by each of the three chosen algorithms and used these as our test points. These two colors were green-yellow and goldenrod which had RGB values of [173, 255, 47] and [255, 193, 37] respectively. We then deployed our algorithm-specific approaches to determine the number of points the adversary must insert in order to change the classification of each color. We varied the size of the training set and for each training set size ran multiple iterations with randomized data. The results of these experiments are summarized in Fig. 3. The y-axis on the plots has been scaled logarithmically due to the large disparity between the algorithms. However, it should be noted that the relationship between the training set size and effort required by the adversary was in fact linear. These experiments allow for a simple comparison between algorithms. For the purpose of detecting anomalous colors, SVMs appear to be significantly more susceptible to adversarial drift than a simple centroid anomaly detector. This also shows the advantage of using a large training dataset for defending against adversarial drift, which must be weighed against the increased cost of acquiring data and training.

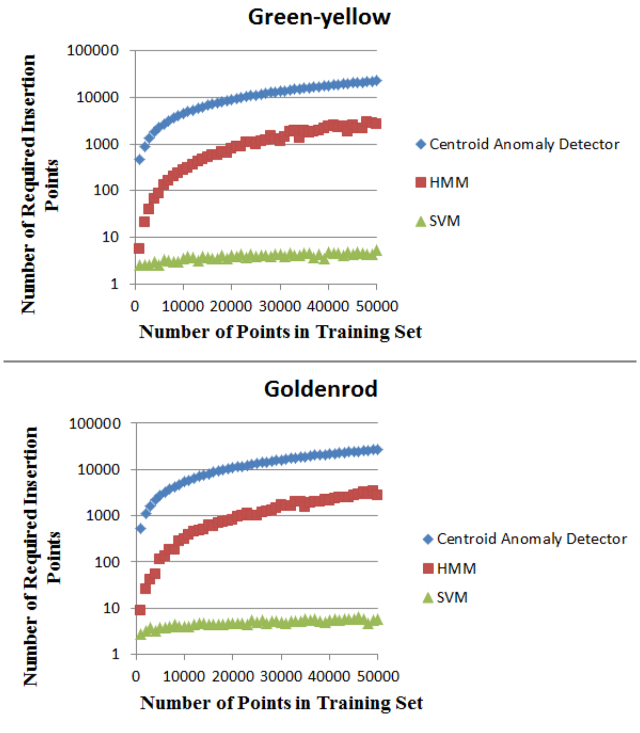


Fig. 3. Effort required by an adversary to cause misclassifications of selected anomalous points using

In the next experiment, for a fixed training set size, we tested each algorithm with a wide variety of different colors to determine the number of points necessary for the adversary to insert in order to force a misclassification on each. This value was then compared against the initial test scores to gain a better understanding of the relationship between the extent to which a point is anomalous and the model's resistance to adversarial drift towards the point. The results of this experiment are summarized in Fig. 4. The centroid anomaly detector and HMM both show a clear correlation between anomaly score and effort required by the adversary. The centroid anomaly detector shows a positive correlation because its test score represents a distance from normalcy, while HMM shows a negative correlation because its test score represents a probability of being benign. The relationship for the SVM is not as clear due to many of the points' initial test scores rounding to zero, but there appears to be a loosely negative correlation. The initial test score is scaled logarithmically (with log(0) set to 0) to make this correlation more clear.

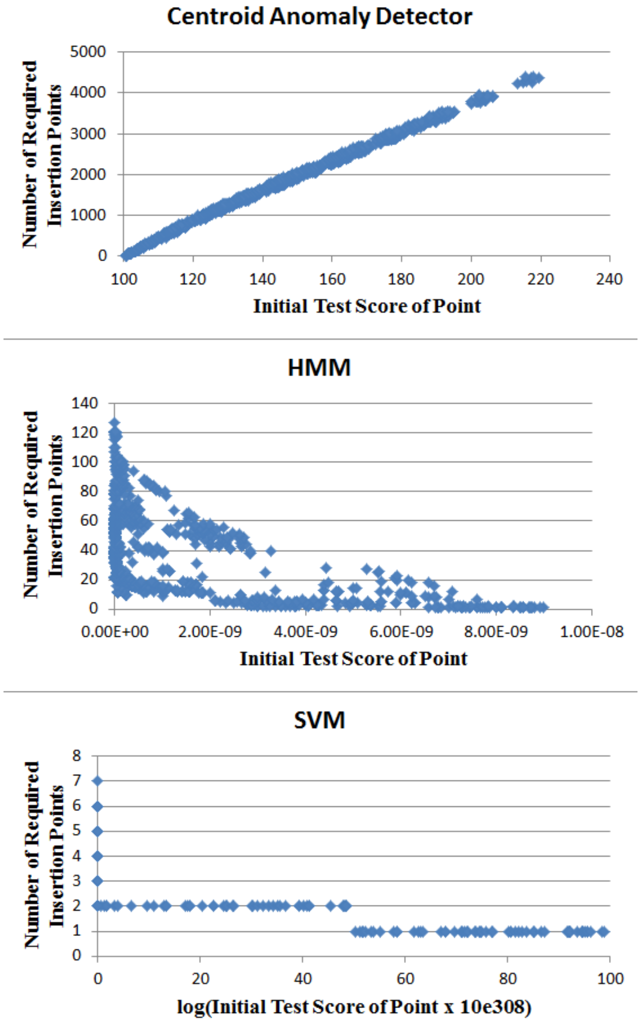


Fig. 4. Effort required by an adversary to cause misclassifications of anomalous points using various

This experimental framework also allows the user to determine values for algorithm specific parameters that are the least susceptible to model drift. For example, Fig. 5 shows the results of four different colors tested against different configurations of the centroid anomaly detector. These configurations represent different methods for setting the threshold value and for adding new points to the training set. The threshold may be set to either a pre-determined fixed value or to a value such that a certain percentage of the training data is considered non-anomalous. Newly received points may either replace a point in the existing training set, retaining a fixed size, or be appended to the end of it. In the plot, the initial test score of the colors, when averaged across varying training set sizes, is compared against the average percent of the size of the original training set that the adversary was forced to inject. This gives a clear indication of which configuration causes the most effort for the adversary. In this instance, setting a fixed threshold value and using an infinite training window would be the most secure.

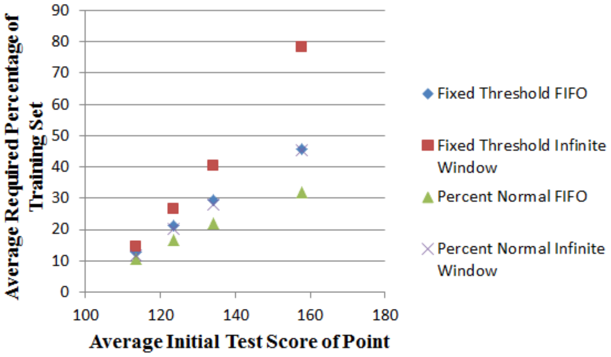


Fig. 5. Effort required by an adversary to cause misclassifications of four anomalous points using four different configurations of the centroid anomaly detector.

The primary purpose of these initial color RGB experiments was to test the legitimacy of the framework's capabilities and to discover baseline patterns. The next step was to validate these results using the open-source IDS HMMPayl with actual network data. Initially, we selected an intrusion point from the identified attack data set that was consistently flagged by HMMPayl when trained with data from the DARPA'99 dataset. This attack point actually consisted of seven packet payloads representing a chunked encoding transfer heap overflow against Microsoft IIS [18]. The HMM-specific adversarial approach was applied until each of the seven payloads went undetected by the IDS. This was repeated multiple times with randomly selected normal traffic for varying training set sizes. The results of this experiment are shown in Fig. 6. A linear relationship between the training set size and the number of insertion points required by the adversary immediately becomes apparent. The slope of the best-fit line reveals that on average the adversary need only insert 0.486% of the training set size to successfully induce model drift while remaining undetected.

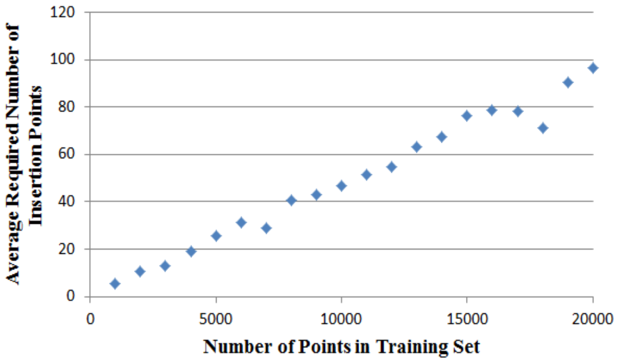


Fig. 6. Effort required by an adversary to cause misclassification of every packet payload in selected attack using HMMPayl

Fig. 7 displays the results of a single run of an attempt to induce model drift on the models created using a training set size of 5000. This shows the impact after each round of retraining that the adversary had on the test score of each of the four initially-flagged payloads from the attack in relation to the anomaly threshold.

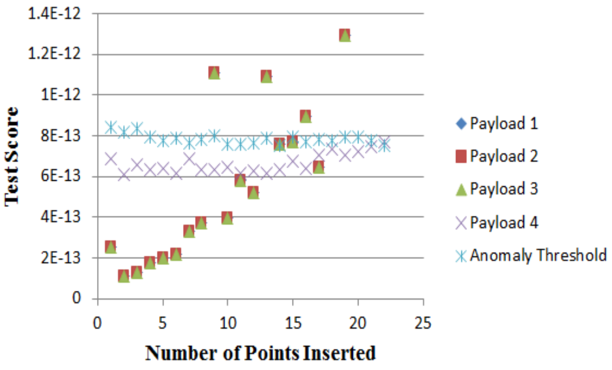


Fig. 7. Resulting HMMPayl test score of payloads from selected attack after the introduction of each insertion point by the adversary

Similar to the color experiments, we next selected every individual packet payload from the attack dataset. For a fixed training set size of 15000 points, we determined the number of insertion points required by the adversary in order to create a misclassification on each payload. This is compared against the initial test score of the payloads to give the security administrator an overall feel for the resiliency of the system. The results of this experiment are summarized in Fig. 8. For the selected attack points there is a loosely negative correlation between the initial test score and the required number of insertion points. It can also be seen that the adversary needs to insert no more than fifteen non-anomalous points, or 0.1% of the training set, in order to create a misclassification.

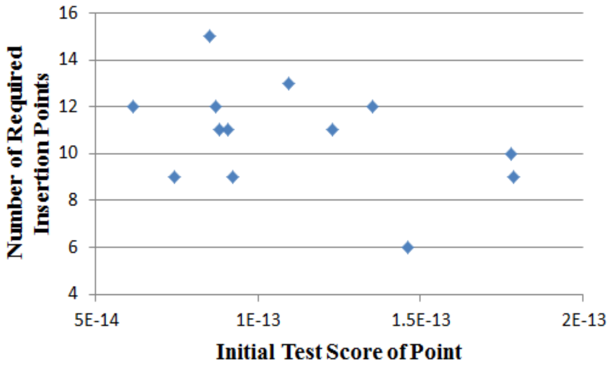


Fig. 8. Effort required by an adversary to cause misclassifications of the selected attack packet payloads using HMMPayl trained with 15000 points.

Conclusions

As a result of the work presented in this paper, we have created a methodology to explore the susceptibility of algorithms used in research-based ML-IDS to induced data drift while they are operating in an adversarial environment. The methodology was developed while examining and subsequently testing several anomaly detectors to establish the baseline approach and results. We further developed and validated the methodology through analysis of additional algorithms implemented in an ML-IDS. We identified potential heuristics to create insertion points in order to induce data drift and isolated a usable method for the HMM-based ML-IDS. We then ran a series of experiments to thoroughly exercise the HMM-based ML-IDS in order to explore it's susceptibility to induced data drift while operating in our tightly controlled adversarial environment. We progressed from the overly-simplified RGB values used to establish baseline results to using the data from the DARPA '99 dataset and the attack dataset from [18] so as to include real-world network traffic data. Our initial experiments demonstrate the type of valuable information that a system administrator may gain through the use of our framework, and preliminary results indicate that the ML algorithms utilized by ML-IDS are indeed susceptible to induced data drift while operating in an adversarial environment.

There are many interesting and novel directions in which the research may progress from this point beyond just further data collection and analysis. Future research should extend to include:

Exploring algorithms not yet covered which have been used or may be used by other ML-IDS [14].

Testing alternative libraries that implement included/excluded algorithms to explore sensitivities related to implementations across identical algorithms.

Investigating additional data types that are considered by IDSs, both commercial and in research, such as trace and log file parsers, executable analyzers and even multi-session analyzers.

Studying defensive remediation that can be used to mitigate the vulnerabilities observed as a part of this work [3].

References

S. Dua and X. Du, Data Mining andMachine Learningin Cybersecurity. Boca Raton, FL: Auerback Publications, 2011.

R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee, “McPAD: a multiple classifier system for accurate payload-based anomaly detection,” in Computer Networks 53, vol. 6, pp. 864-881, 2009.

M. Barreno, B. A. Nelson, A. D. Joseph, and J. Tygar, “The security of machine learning,” in Machine Learning, vol. 81, no. 2, 2010, pp. 121-148.

A.Kantchelian, et al., "Approaches to Adversarial Drift", in AISec’13, New York: ACM, 2013, pp. 99-110.

B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G.Giacinto, and F. Roli, “Evasion atacks against machine learning at test time,” in Machine Learning and Knowledge Discovery in Databases, Springer Berlin Heidelberg, 2013, pp. 387-402.

B. Biggio, B. Nelson, and P. Laskov, “Poisoning attacks against support vector machines,” in arXiv preprint arXiv, vol. 1206, no. 6389, 2012.

M. Kermani, S. Sur-Kolay, A. Raghunathan, N. Jha, "Systematic Poisoning Attacks on and Defenses for Machine Learning in Healthcare" in IEEE Journal of Biomedical and Health Informatics, vol. PP, New York: IEEE, 2014, pp. 1.

K. Tan, K. Killourhy, R. Maxion, "Undermining an Anomaly-Based Intrusion Detection System Using Common Exploits", in RAID'02 Proceedings of the 5th international conference on Recent advances in intrusion detection. Heidelberg: Springer-Verlag, 2002, pp. 54-73.

D. Ariu, R. Tronci, and G. Giancinto, “HMMPayl: an intrusion detetion system based on hidden markov models,” in Computers and Security 30, no. 221, 2011.

B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. IP Rubenstein, U. Saini, C. A. Sutton, J. D. Tygar, and K. Xia, “Exploiting machine learning to subvert your spam filter,” in LEET 8, 2008, pp.1-9.

K. Nelson, G. Corbin, M. Blowers, “Evaluating data distribution and drift vulnerabilities of machine learning algorithms in secure and adversarial environments,” in SPIE Sensing Technology+ Applications, International Society for Optics and Photonic, 2014.

M. Kloft and P. Laskov, “Online anomaly detection under adversarial impact,” 2011.

B. Scholkopf, J. Platt, J. Shawe-Taylor, A. Smola, R. Williamson, "Estimating the Support of a High-Dimensional Distribution", in Technical Report MSR-TR-99-87, Redmond: Microsoft Research, 2000.

C. Tsai, Y. Hsu, C. Lin, W. Lin, "Intrusion detection by machine learning: a review", in Expert Systems with Applications, Vol. 36, Elsevier, 2009, pp. 11994–12000.

I. Corona, R. Tronci, G. Giacinto, "SuStorID: A multiple classifier system for the protection of web services", Pattern Recognition (ICPR), 2012 21st International Conference, IEEE, 2012 pp. 2375-2378.

Rabiner, L., "A tutorial on hidden Markov models and selected applications in speech recognition," Proceedings of the IEEE , vol.77, no.2, pp.257,286, Feb 1989

R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, “The 1999 DARPA off-line intrusion detection evaluation,” in Computer networks 34, no. 4, 2000, pp. 579-595.

K. L. Ingham, and H. Inoue, “Comparing anomaly detection techniques for http,” in Recent Advances in Intrusion Detection, Springer Berlin Heidelberg, 2007, pp. 42-62.

M. V. Mahoney, and P. K. Chan, “An analysis of the 1999 DARPA/Lincoln laboratory evaulation data for network anomaly detection,” in Recent Advances in Intrusion Detection, pp. 220-237, Springer Berlin Heidelberg, 2003

Bibliography

S. Dua and X. Du, Data Mining and Machine Learning in Cybersecurity. Boca Raton, FL: Auerback Publications, 2011.

R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee, “McPAD: a multiple classifier system for accurate payload-based anomaly detection,” in Computer Networks 53, vol. 6, pp. 864-881, 2009.

M. Barreno, B. A. Nelson, A. D. Joseph, and J. Tygar, “The security of machine learning,” in Machine Learning, vol. 81, no. 2, 2010, pp. 121-148.

A.Kantchelian, et al., "Approaches to Adversarial Drift", in AISec’13, New York: ACM, 2013, pp. 99-110.

B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G.Giacinto, and F. Roli, “Evasion attacks against machine learning at test time,” in Machine Learning and Knowledge Discovery in Databases, Springer Berlin Heidelberg, 2013, pp. 387-402.

B. Biggio, B. Nelson, and P. Laskov, “Poisoning attacks against support vector machines,” in arXiv preprint arXiv, vol. 1206, no. 6389, 2012.

M. Kermani, S. Sur-Kolay, A. Raghunathan, N. Jha, "Systematic Poisoning Attacks on and Defenses for Machine Learning in Healthcare" in IEEE Journal of Biomedical and Health Informatics, vol. PP, New York: IEEE, 2014, pp. 1.

K. Tan, K. Killourhy, R. Maxion, "Undermining an Anomaly-Based Intrusion Detection System Using Common Exploits", in RAID'02 Proceedings of the 5th international conference on Recent advances in intrusion detection. Heidelberg: Springer-Verlag, 2002, pp. 54-73.

D. Ariu, R. Tronci, and G. Giancinto, “HMMPayl: an intrusion detection system based on Hidden Markov Models,” in Computers and Security 30, no. 221, 2011.

B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. IP Rubenstein, U. Saini, C. A. Sutton, J. D. Tygar, and K. Xia, “Exploiting machine learning to subvert your spam filter,” in LEET 8, 2008, pp.1-9.

M. Kloft and P. Laskov, “Online anomaly detection under adversarial impact,” 2011.

B. Scholkopf, J. Platt, J. Shawe-Taylor, A. Smola, R. Williamson, "Estimating the Support of a High-Dimensional Distribution", in Technical Report MSR-TR-99-87, Redmond: Microsoft Research, 2000.

C. Tsai, Y. Hsu, C. Lin, W. Lin, "Intrusion detection by machine learning: a review", in Expert Systems with Applications, Vol. 36, Elsevier, 2009, pp. 11994–12000.

I. Corona, R. Tronci, G. Giacinto, "SuStorID: A multiple classifier system for the protection of web services", Pattern Recognition (ICPR), 2012 21st International Conference, IEEE, 2012 pp. 2375-2378.

Rabiner, L., "A tutorial on hidden Markov models and selected applications in speech recognition," Proceedings of the IEEE , vol.77, no.2, pp.257,286, Feb 1989

R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, “The 1999 DARPA off-line intrusion detection evaluation,” in Computer networks 34, no. 4, 2000, pp. 579-595.

K. L. Ingham, and H. Inoue, “Comparing anomaly detection techniques for http,” in Recent Advances in Intrusion Detection, Springer Berlin Heidelberg, 2007, pp. 42-62.

M. V. Mahoney, and P. K. Chan, “An analysis of the 1999 DARPA/Lincoln laboratory evaulation data for network anomaly detection,” in Recent Advances in Intrusion Detection, pp. 220-237, Springer Berlin Heidelberg, 2003.

Tavallaee, M., Stakhanova, N., & Ghorbani, A. A. (2010, September). Toward Credible Evaluation of Anomaly-Based Intrusion-Detection Methods. *IEEE Transactions on Systems, Man, and Cybernetics - Part C, 40*(5), 516-524.

LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

AFFG	Adaptive Fuzzy Fitness Granulation
AFRL	Air Force Research Laboratory
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BSM	Basic Security Module
DARPA	Defense Advanced Research Projects Agency
DB	Database
DoD	Department of Defense
DT	Decision Tree
GA	Genetic Algorithm
GUI	Graphical User Interface
HMM	Hidden Markov Model
HPC	High Performance Computer
HTTP	Hypertext Transfer Protocol
ICS	Industrial Control Systems
IDS	Intrusion Detection System
IMAP	Internet Message Access Protocol
IO	Input/Output
JDBC	Java Database Connectivity technology
JMS	Java Message Service
KDD	Knowledge Discovery and Data Mining
KNN	K-Nearest Neighbor
LDAP	Lightweight Directory Access Protocol
ML	Machine Learning
ML-IDS	Machine Learning-based Intrusion Detection System
MOM	Message Oriented Middleware
OISF	Open Information Security Foundation
PCA	Principal Component Analysis
PCap	network Packet Capture
POP3	Post Office Protocol 3

PSO	Particle Swarm Optimization
RBF	Radial Basis Function
REST	REpresentational State Transfer
RGB	Red Green Blue
RMI	Remote Method Invocation
SA	Simulated Annealing
SCADA	Supervisory Control And Data Acquisition
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SOM	Self-Organizing Map
SVM	Support Vector Machine
TCP	Transmission Control Protocol
VM	Virtual Machine
XML	Extensible Markup Language