2014

Unsupervised Learning Trojan

Arturo Geigel
Nova Southeastern University, ag1011@nova.edu

Unsupervised Learning Trojan

by

Arturo Geigel

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
Computer Science

Graduate School of Computer and Information Sciences
Nova Southeastern University
2014

We hereby certify that this dissertation, submitted by Arturo Geigel, conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

Dr. Wei Li, Ph.D. Chairperson of Dissertation Committee	Date
Sumitra Mukherjee, Ph.D. Dissertation Committee Member	Date
James D. Cannady, Ph.D. Dissertation Committee Member	Date

Approved:

Eric S. Ackerman, Ph.D. Dean, Graduate School of Computer and Information Sciences	Date

Graduate School of Computer and Information Sciences
Nova Southeastern University

2014

An Abstract of a Dissertation Submitted to Nova Southeastern University
in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Unsupervised Learning Trojan

by

Arturo Geigel

This work presents a proof of concept of an Unsupervised Learning Trojan. The Unsupervised Learning Trojan presents new challenges over previous work on the Neural network Trojan, since the attacker does not control most of the environment. The current work will presented an analysis of how the attack can be successful by proposing new assumptions under which the attack can become a viable one. A general analysis of how the compromise can be theoretically supported is presented, providing enough background for practical implementation development. The analysis was carried out using 3 selected algorithms that can cover a wide variety of circumstances of unsupervised learning. A selection of 4 encoding schemes on 4 datasets were chosen to represent actual scenarios under which the Trojan compromise might be targeted. A detailed procedure is presented to demonstrate the attack's viability under assumed circumstances. Two tests of hypothesis concerning the experimental setup were carried out which yielded acceptance of the null hypothesis. Further discussion is contemplated on various aspects of actual implementation issues and real world scenarios where this attack might be contemplated.

Acknowledgments

I would like to thank my wife Gina and my parents for providing the support to keep me going throughout my career and studies. I will also like to dedicate this work to my son Ryan, which came to life during this dissertation process and has made my life richer.

I will like to thank the committee members, Dr. Wei Li, Dr. Sumitra Mukherjee and Dr. James Cannady for their support and guidance throughout the dissertation process. The input provided throughout the process helped me to focus, and look at the research in ways that I would normally miss. This experience will be with me for the rest of my life, hopefully making me a better scholar.

I would also like to thank Dr. Kay Berkling who was also my mentor during my Masters degree, guided me, and provided the inspiration to pursue doctoral studies.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Background

Machine learning has become a mainstream tool for analysis in various industries. Among the applications of machine learning are financial applications (Krollner et al., 2010; McNelis, 2005), medical applications (Kononenko, 2001), and intrusion detection (Tsai et al., 2009; Sabhnani & Serpen, 2003a), among many others. While the field is blossoming rapidly, there are possible attacks that can target the application of such tools. During the last few years, researchers have noticed subversion attacks to the underlying machine learning algorithm. Such attacks are commonly known under the umbrella of adversarial knowledge discovery (Skillicorn, 2009) or adversarial learning (Lowd & Meek, 2005). The attacks are successful because in most cases the algorithms are designed under the assumption that the underlying distribution is static or (very) slowly varying over time.

Most machine learning algorithms are designed to generalize the behavior of the distribution given a sample of the population. The specific procedure of generalization will depend on the underlying algorithm, and if the algorithm utilizes supervised or unsupervised learning. Once the algorithm is coded with the underlying assumptions, its data generalization abilities are embedded into the system, and depending on its robustness, it will respond to changes in the distribution with a better or worse performance. These assumptions can be exploited to the benefit of an attacker that has knowledge of the underlying mechanisms of the algorithm. The current problem is that this analysis has only recently begun to take place in the adversarial knowledge discovery community to address this type of malicious attacks.

This work presents a new attack on unsupervised algorithms based on the Neural network Trojan (NnT) proposed by Geigel (2013). The new attack poses some new challenges to the attacker, since some of the control leveraged in the NnT's supervised learning cannot be harnessed in an unsupervised scenario.

This chapter first presents the problem statement which provides information on why such attacks must be studied and analyzed to provide better security in machine learning algorithms. The presentation follows with the dissertation goal that delineates the objective of building a proof of concept of such attack and to document its procedure. The dissertation goal is followed by the research questions which this work will investigate while pursuing the study of the Unsupervised Learning Trojan (ULT). The section that follows elaborates on the significance of the Unsupervised Learning Trojan and why its study should be pursued. The barriers and issues will be presented to document the challenges on carrying out such compromise. The challenges provide enough information to elaborate the approach used to analyze the Unsupervised Learning Trojan and how to carry out the proof of concept of such an compromise.. Finally, the proposal finishes with the reference section.

Problem Statement

This section introduces the main problem of study, which is the *Unsupervised Learning Trojan density problem* and its sub-problem of cluster identification. Before details of the problem are given, some background is provided on how the problem arises by discussing the areas of spam and intrusion detection, which are active areas of adversarial machine learning research. Within this area, a closer look is given to the problem of unsupervised learning which is just beginning to be studied.

With this background, the problem of adversarial learning in unsupervised scenarios is generalized to encompass a field that is greater than just intrusion detection, but also encompasses all applications containing unsupervised learning algorithms by implementing the ULT. It is within this field of reference that the *Unsupervised Learning Trojan density problem* and its sub-problem of the cluster identification are framed in the discussion that follows.

Unwanted commercial messages (UCM) and malware and can be considered tools in furthering the underground economy. The use of UCM or spam makes a significant part of the Internet messages, and also to the underground economy in the form of scams and compromises (Stone-Gross et al., 2011). By the same token, malware in general is used in the underground economy as a tool to obtain stolen credentials (Holz et al., 2009). UCM and malware can be said to share two common characteristics: the first is that they are motivated by economic trends, and the second is that the techniques to target the intended recipient are changing as countermeasures evolve.

The rise of computer malware has been, to a large extent, an evolutionary process that started in the 1980s, and there is information that it has evolved from minor activities of nuisance to profit oriented attacks (Hughes & DeLone, 2007; Moore et al., 2009; Simion, 2010). This profit based driven activity has also been seen in UCM (Plice et al., 2008). To make these activities profitable, they have to overcome the hurdles of filtering mechanisms that dampen the effects of target acquisition. In the case of the UCM, the target is the potential buyer; and in the case of a computer compromise, the target may be a machine or the assets stored in the machine.

Countermeasures to filter the UCM and the malware threat have evolved in response to the changing nature of the tactics of the attackers and spammers. These, in turn, have made the attacks more sophisticated. Some of this sophistication developed in the form of cryptography in viruses (Young & Yung, 1996; Abidin et al., 2012). Other fairly recent methods have been more targeted attacks in the form of advanced persistent threats (APT) (Jeun et al., 2012). The degree of sophistication of malware in general has led to the development of more evolved intrusion detection techniques such as those using machine learning techniques. These, in turn, have also led to malware developers and researchers to explore the strengths and weaknesses of these systems. Research involving vulnerabilities in machine learning has been brought to light recently in the literature (Barreno et al., 2008, 2010). Some specific examples of the possible impact of this analysis are shown in the successful evasion of machine learning algorithms such as SpamBayes (Nelson et al., 2008, 2009). Under this scenario, the objective is to subvert the training of a spam filter. The authors classify the attacks on spam filters into two categories: causative availability attacks and causative integrity attacks. Under causative availability, the objective is to classify ham (benign) messages as spam. This means that benign messages will fall as undesired messages. A contemplated scenario of this attack may be a spammer that wants to force a user to look for his good emails in the spam category, forcing the user to look at the spam message folder, and therefore achieving his goal. A second scenario under this attack is a malicious contractor who would focus the spam to deny access to other competitors by sending spam messages with words and labels used by his competitors. In contrast, under causative integrity attacks, the objective is to create pseudospam where spam message content can be taken from news articles or other trustworthy source; however, they also contain spam tokens so as to shift the training to accept some carefully crafted messages.

Machine learning subversion has been mostly directed towards intrusion detection and spam, where misdirection to bypass a filtering mechanism is the goal. The analysis of such attacks, especially unsupervised learning algorithms, is of a nontrivial nature. A general analysis of unsupervised machine learning subversion was started in the work of Nelson & Joseph (2006). In this work, the objective was to carry out an attack by poisoning the data input to the unsupervised learning classifier. The attack's focus is to move the centroid progressively until the radius of the hypersphere that characterizes the benign data falls within the boundary of the attack. This allows the attack to pass as benign data. Under the scenario proposed by Nelson & Joseph, the attacker would require an exponential number of attack points to carry out an effective attack on the algorithm. In other words, it requires exponential effort to displace the center of mass as it grows with the input data.

The analysis of the attack was extended in Kloft & Laskov (2012), in which the assumption of infinite horizon window where the algorithm attacked memorizes all the data seen so far is relaxed towards more realistic online algorithms that might be subjected to attacks. Their analysis focuses on online algorithms that discard data points, and it was found that recalculating the center of mass maintaining n data points needs only linear amount of injected points. This analysis also held for the *random out* selection of data points where a data point is selected at random and is removed. However, for the *nearest* *out* strategy, the nearest neighbor of the new data point is removed, the complexity depends on the Voronoi cell sizes which are that sets of points which lie closer together than any other set of points created by the data. The analysis carried out by Kloft & Laskov highlights that real scenarios attacks on unsupervised learning are feasible to carry out under reasonable realistic assumptions.

In Battista et al. (in press), the authors identified an additional type of attack using obfuscation. As opposed to poisoning the data stream, obfuscation aims to hide attack samples into an already existing cluster by manipulating feature values. The attack's objective is to insert the malicious data into the cluster while avoiding significant alterations on the clusters.

Another type of attack against machine learning is in the form of Trojan payloads encoded in the data. The first Trojan of this type is the Neural network Trojan (NnT) presented by Geigel (2013). The work showed that by also contemplating a compromise on the algorithm implementation, it is possible to hide a payload in the encoded data stream fed to a learning neural network. The principal challenge in this type of exploit is the learning and generalization capabilities that are inherent in a supervised learning algorithm. To avoid detection, the generalization, memorization and sequential execution of the payload of the Trojan properties of the algorithm must be present, to achieve the desired exploit. Another challenge is to select the appropriate encoding that favors both the benign data, as well as the payload selection. In the NnT the payload selection hinged on a minimal payload that did have the least amount of recurring strings, such that the memorization could be successfully implemented while still retaining most of the generalization properties for the intended data set.

The broader question of whether the analysis of the NnT can be utilized in other machine learning techniques remains an open question, and the implications for it can be substantial. One specific area where this could be important is in the financial sector, where machine learning is contributing substantially in the form of high frequency trading using unsupervised learning algorithms such as Self Organizing Feature Maps (SOFM) (Resta, 2009; Blazejewski & Coggins, 2004) and K-means (Ai et al., 2010; Ye et al., 2011).

In trying to address whether unsupervised learning can be compromised to carry a Trojan payload, several challenges must be addressed. These challenges on the part of the adversary are very different in nature than those present in poisoning and obfuscation attacks against unsupervised learning algorithms. The unsupervised learning Trojan has to deal with discrimination of the Trojan payload against the benign data. This means that in order to trigger the attack, the clustering algorithm must segregate an actual attack from the rest of the data. This specific problem of the attack will be referred to as the *Unsupervised Learning Trojan density problem.* A sub-problem of the Trojan density problem is that of the *cluster identification* as a malicious cluster within those clusters identified by the algorithm. Both problems presents particular implementation differences depending on whether the number of clusters is fixed or dynamically determined. Another challenge of the exploit deals with the interpretation of the cluster points and the encoding/decoding of the payload within the cluster elements. This falls outside of the machine learning attack and deals with how the data points are encoded in a feasible encoding/decoding sequence for execution.

Dissertation Goal

The goal of the dissertation was to present the arguments and a proof of concept that a Trojan can be inserted in unsupervised learning algorithms. Its aim was to expand the analysis of the learning vulnerabilities found in the NnT to a scenario in which unsupervised learning is compromised. The analysis of the Unsupervised Learning Trojan differs substantially from the NnT, and it presented new challenges for a successful compromise. The work tested the ULT on several unsupervised learning algorithms to show different techniques and assumptions the attacker must make to carry out the attack. The attack can be carried out by compromising the programming and afterwards inserting malicious elements into the data stream to be processed by the algorithm.

The proposed work for the ULT consisted of analyzing the steps necessary to carry out the compromise. The analysis concentrated on a general framework where the attacker methodologically tested the algorithm before launching the actual exploit. This methodology resembles the experimental setup of training and testing supervised learning algorithms. This exercise helped to determine the effectiveness of the attack on three specific unsupervised learning algorithms, but at the same time give enough information to generalize the compromise to other unsupervised algorithms. By analyzing the methodology carried out by an attacker, security professionals can analyze the behavior of the exploit, and therefore allow for the possibility of finding successful defenses to such attacks.

Research Questions

Throughout the research carried out in this dissertation, several questions were contemplated that were part of the analysis of the ULT.

- Can the Trojan attack be carried out successfully on the proposed algorithms?
- Could the compromised algorithms present symptoms of compromise without analyzing the code?
- What are the strategies for attack hiding?
- Do the algorithms require many modifications to carry out the attack?

Relevance and Significance

The attacks on machine learning algorithms through Trojans were first proposed in Geigel (2013), and present a novel attack surface that has not been previously explored. The capacity for encoding the attack on data streams of information for the learning algorithm presents new challenges in analysis and detection. As covered in the NnT, the probability of substantial polymorphism is a reality with Trojan embedding on a machine learning algorithm. What differentiates this type of Trojan from previous work is that the nature of success and hiding depends on the statistical properties of the data and the algorithm, as opposed to just programming vulnerabilities. The current work aims at extending the scope of the analysis to unsupervised learning, and to highlight the unattended dangers of Trojans on these types of algorithms. The impact on these types of compromises extend to any application that uses the algorithms and may include applications in the security, finance, health care fields, and unmanned vehicles, among other fields. By exposing these vulnerabilities, proper steps can be taken to protect the algorithms and prevent the detrimental effects of such attacks.

Studying these types of Trojans in machine learning algorithms and disseminating this information to the security community can have a positive effect in securing these types of algorithms from such attacks (Nizovtsev & Thursby, 2007; Arora et al., 2010). The proposed work will present a detailed analysis of the attack mechanisms used in carrying out this type of compromise and the methodology used during these attacks, such that future work can concentrate on detection and avoidance before actual threats emerge. The presentation of the ULT will expand on the work of Geigel (2013), which only dealt with a specific type of neural network. By contrast, the present work will generalize Trojan attacks in unsupervised machine learning. The generalization to unsupervised learning is a step further towards a future analysis: that in principle all machine learning algorithms could be susceptible to Trojan attacks.

Barriers and Issues

The challenge with ULT implementations is that there is no training set as in the NnT. This makes the attack concept more difficult to carry out since the attacker does not control the distributions used for the learning. Nonetheless, the attacker is assumed to have access to partial capture of the data stream that will be passed to the unsupervised learning. To demonstrate the attacks viability, it must be shown that the attacker can select the payload of the Trojan in a manner that maximizes the probability of success in carrying the attack.

The execution of such probability maximization is non trivial and the proposed work will focus on the techniques that an attacker would have to develop in order to carry out the compromise.

Assumptions, Limitations, and Delimitations

One of the challenges evident in the field of study of compromises is that it is difficult to frame what an adversary will carry out within the scope of a compromise. The nature of compromises imposes few restrictions on the field of study by its very nature and several strong assumptions must be made to delimit the scope. This scope delimitation, in turn, might not necessarily reflect the actual decisions and scenarios which the attacker may choose. A very strong underlying assumption made in this study is that the attacker will be limited to attacking the learning algorithm directly, that is having access to the source code. Under real world scenarios the adversary will use any technique deemed effective in achieving its objective which may be very simple or complex in nature. Notwithstanding, the delimitation of this study to techniques used to compromise just the field of unsupervised learning must be assumed to delimit the scope of this study to an achievable one.

Another delimitation is that the attack optimization will be done based on abstract grounds and will not take into account domain knowledge. While such knowledge would certainly help both the attacker and the system designer, it would take away any generalization that can be made from this study. The compromise can also be realized by an attacker that can insert the code into the algorithm through access to the source code or into executable files via code caves (Gawkowski & Smulko, 2010), however this stage of the attack will be assumed and not be explored here due to the amount of possible scenarios to explore that fall outside adversarial machine learning field. The study will focus on the actual implementation of the data insertion and algorithm interpretation of the exploit. Within the scope of unsupervised learning algorithms, the breadth of this study must also be further demarcated to be manageable, but at the same time provide enough information on the possibility that the vulnerability is general enough to cover the field. This study will be delimited in scope in: the algorithms chosen, the type of compromise on each algorithm, the data selection, and the payload executed on each of the algorithms chosen.

The algorithms' selection has been delimited to the following: K-means algorithm, Kohonen Neural Network, and Adaptive Resonance. While the field provides an extensive number of algorithms such as: spectral clustering, hierarchical clustering, and mixture models (Webb, 2011), the chosen algorithms reflect some of the general behaviors of the unsupervised learning field.

The use of the data sets from the UCI machine Learning Repository (Asuncion & Newman, n.d.) provide a recognized source of data that has been submitted for peer review and at the same time available for test reproduction purposes. The selection of specific data sets was based on possible scenarios on which the adversary might deploy the actual compromise. The objective is to provide a simulation that will closely resemble that of an actual compromise on a real world system. The data sets were constrained in quantity in an attempt to maximize the effort to show as many compromises on specific data sets, as opposed to just one compromise on many data sets.

Selection of payloads was done to provide a executable payload which might be harder to conceal than other types of attacks such as ASCII payloads or simple operating system commands. The selection of payloads varied in size and specific targets to demonstrate versatility of the compromise.

Definition of Terms

The following provides a list of specific terms used throughout this work.

Calibration phase - A step in the attacker's design of the compromise that tests the assumptions of the attack before the actual compromise takes place.

Cluster Identification Sub Problem - The identification of malicious clusters within a group of clusters provided to a algorithm.

Compromising Hull - A collection of malicious points that make up the external “hull” or outline of points that enclose the malicious cluster center.

Degree of similarity (DS) - A measure used to determine the degree of overlap between the clusters, by measuring how many points of a data set were included in a cluster, other than the expected one.

Degree of Cluster Distance (DCD) - The ratio between cluster distances and the spread of the data belonging to each cluster.

Density Estimation - An estimate of an underlying probability density based on a sample of empirical data obtained from the underlying distribution.

Embedding Problem - The problem of inserting a malicious data set within a normal training set, with the objective of achieving a compromise.

Extrinsic Property - A cluster property derived from the comparison of one cluster to other clusters.

Hypersphere - An abstract representation of the generalization of a sphere on higher dimensional space.

Imbalanced Data Set - A data set that contains a majority of samples belonging to a specific category or distribution interval and therefore skews the representation of the underlying distribution of data or category labels.

Intrinsic Property - A comparison of the cluster’s member elements amongst themselves to derive a common property that describes the cluster.

Mixture of Densities - The probability density of more than one unrelated data set. Specifically, it refers to the combined probability density of one or more benign data sets with that of a malicious data set.

Poisoning - The process of inserting malicious data points into a data set with the objective of misleading the algorithm evaluating the data.

Polymorphism - The capacity of a malware to change or mutate its appearance to avoid detection, while at the same time preserving the capacity to compromise a host and carry out the compromise.

Unified Distance Matrix (U-Matrix) - is a representation of the Kohonen network that is measured using the average distance between a node's four immediate neighbors.

Summary

The field of adversarial machine learning is becoming an increasingly important topic in machine learning, and will become more important as more applications in security, as well additional critical applications outside the field of security adopt machine learning techniques. Machine learning has recently been expanded to include an area of research that deals with machine learning under adversarial attack. Under these scenarios, most machine learning algorithms fail, since they assume a static underlying distribution and cannot handle an active adversary that manipulates the data. A closely related adversarial attack is that of inserting malware leveraging machine learning characteristics such as the Neural network Trojan and the Unsupervised Learning Trojan.

The unsupervised learning Trojan is a malware compromise designed to take advantage of an application that uses unsupervised learning algorithms. The attack's complexity is higher than that of the neural network Trojan in that the attacker does not have a controlled environment through a training set on which to supply predetermined data to the algorithm. Instead, the attacker must estimate the effects of the data on the attack's effectiveness, and based on the testing scenarios, he can estimate the probability of success.

Chapter 2

Review of the Literature

Introduction

Until recently, most of the work on the security of machine learning focused on using the learning algorithm to learn patterns to detect malicious activity. These type of intrusion detection systems are based on anomaly detection (Verwoerd & Hunt, 2002) and misuse detection (Sabhnani & Serpen, 2003b). The research into compromises as part of the machine learning field have been generated as part of a new field known as adversarial machine learning or adversarial discovery (Skillicorn, 2009). These methods of subverting machine learning algorithms are closely related to other methodologies, such as Trojans, whose objective is the subversion of the machine by masquerading itself as an innocent application. Such hiding methods are also employed in cryptovirology, where the executable code is partly hidden by encryption or it uses encryption to hijack data. All these recent trends lead to a new field of study of machine learning for security, but also to a field in sophisticated malware techniques that need to be studied if there is to be protection from them.

Trojans

Arriving at a formal definition of a term in malware research is difficult, since the field is being redefined constantly by new types of malware which have different characteristics. The definition of Trojan is no exception. The use of the word Trojan in malware has its origins on the Greek tale of Odysseus that had the idea of hiding Greek soldiers in a wooden horse, in order to infiltrate Troy and win the war. An analogy of the technique evolved by the similarity of the methods used to hide a malicious payload in computers. Since the translation to computer systems is done by analogy, it depends on what features particular author is focusing on to characterize the Trojan attack. This contributes to a lack of consensus on what constitutes a Trojan, and makes it difficult to find relevant literature which might be classified under other topics, but might be relevant to the actual research being conducted. Another justification for studying the categorization, lies in the high false or failure alarm rates of current behavior analysis techniques based on Trojan detection strategies (Chen et al., 2009). A brief survey on the different approaches the previous literature has taken towards clarifying the definition will help in framing the discussion further.

Whalley (1999) covers various definitions of Trojans that appear in the literature and arrives at the conclusion that the definitions are all lacking in one way or another. The author presents his definition as:

> "A program which the user thinks or believes will do one thing (the 'perceived purpose'), and which may or may not do that thing, but which also does something else which is not necessary to accomplish the perceived purpose, and of which the user would not approve (the 'payload')."

The definition tries to overcome the shortcomings of the previous definitions by avoiding intent, purpose and malicious outcome usually mentioned in the previous definitions. Notwithstanding, Whalley also concedes that his alternative definition also poses some drawbacks by avoiding specificity and intent which may make other types of programs fall under the category of Trojan.

Thimbleby et al. (1998) provide the basic components of viral infection in which the Trojan component is included. The authors also put forth that Turing machines are inadequate to address the characterizations of infections. This is also recognized to some extent by F. Cohen (2001). One of the objections of using Turing machines that specifically pertain to Trojans is the problem of framing what is masquerading. Masquerading is defined in Thimbleby et al. as the notion "where a user knows the names of programs and anticipates their likely behavior". Another difficulty with Turing machine models is defining the notion of "other" programs, as is the case of a Trojan payload. The authors provide an alternative approach to Turing machine models by specifying a *representation*, which represents the full state of a machine and an *environment map* that can be taken to be the configuration of the system. Finally, a *meaning* represents what a program does when it is executed in the machine, and the definitions of *similarity* and the *for most* quantifier clarifies what the usual term of 'indistinguishable' means. This marks an important development that defines the environment where Trojan executes and its characteristics, since without the distinguishable/indistinguishable property one cannot identify programs running in the model as Trojan or non-Trojan.

The work of Thimbleby et al. was objected by Mäkinen (2001) on the grounds that the five arguments exposed by Thimbleby et al. are not enough to substitute Turing machine models. In a rebuttal, Thimbleby et al. (2001) explain that alternatives such as those exposed by Mäkinen would entail intricate solutions. One of the intricate solutions would imply the invocation of an environment, observer and the notion of 'true meaning'. Thimbleby et al. do recognize that both solutions still need to be further explored, in order to settle the matter.

In Zuo et al. (2006), the authors take a different approach to F. Cohen (1987, 2001) and Thimbleby et al. (1998), and extend (Adleman, 1990; Zuo & Zhou, 2004) concept's of recursive functions to describe the notion of infection and imitation within this framework. The imitation property consist of a recursive program that behaves in some computation like the original program by defining it in terms of the existential quantifier. This allows for variation on some computations, while behaving differently in others, and hence, this allows for the notion of imitation which is one of the core characteristics of a Trojan.

An alternative to defining a Trojan horse is to use a general taxonomy under which, if the software contains N characteristics then it can be considered to be infected by a Trojan, and therefore be compromised. This approach is taken in (Karresand, 2003). In his work, the author uses the technical description model of IT weapons (TEBIT) taxonomy. The objective with this approach is to find a representative group of characteristics of the software weapons to be defined, categorize them, and calculate the standard deviation from the main characteristics.

While the problem of defining a Trojan and its characteristics takes a substantial amount of literature, there is a segment of the literature that avoids the problem altogether and follows a more operational approach, focusing on the detection of the Trojans based on assumed characteristics or behavior. Qin et al. (2010) use a module that detects operating systems objects such as registry, file, and ports systems services, among others, to monitor the state of the operating system. Each object being monitored is assigned a weight and a suspicious degree, and is then evaluated using a dot product; and if it exceeds a certain threshold value, then it is labeled as a Trojan. Tang (2009) focuses on the Microsoft PE executable format, uses decision trees to derive relevant characteristics of the format, and uses these filtered characteristics to train a neural network. The experimental results show that the selected headers of the PE executable format provide better results than using all headers. Alternatively, Y.-F. Liu et al. (2010) use instance based learner, Naive Bayes, and decision trees to evaluate a system's catalog of operating system behavior. The system behavior catalog includes multiple attributes in network messages, processor, thread, system, and object categories. The authors concluded that depending on the specific mixture of attributes, some might become relevant, while others might become less relevant and can provide lower results. Further, the authors found that NaiveBayes performed better in terms of efficiency and accuracy than the other compared classifiers.

A recent trend in malware is the addition of Trojans at the hardware level. A proposed taxonomy to describe hardware Trojans is given in Tehranipoor & Koushanfar (2010). The authors' taxonomy focuses more on hardware mechanisms such as particular activation characteristics actions and physical characteristics that distinguishes hardware Trojans. While some are related to software Trojans, other mechanisms, such as activations through sensors or logic circuit execution, are particular to hardware Trojans.

Baumgarten et al. (2011), on the other hand, put particular interest on the hardware Trojans distribution channel, which is particularly vulnerable to attacks. They describe the vulnerabilities in hardware distribution channel and how trust, lack of secure part authentication, and theft can all play to the attacker's advantage when using hardware Trojans. The authors also describe different types of attackers along the supply chain: designer attackers, fabrication attackers, and distribution attackers.

A particular hardware Trojan horse that has been documented, which is of particular significance, is given in Wei & Potkonjak (2013). The Trojan is particular in its difficulty in detection, since it relies on stressing certain gates and delay faults within the integrated circuit's output. These methods of hardware Trojan activation due to stress can be quite effectively implemented, based on the available documentation and represent a real concern in modern circuit design.

A field of virology that is not directly related to Trojans, but has been used in Trojans, has been Cryptovirology (Young & Yung, 2004). Cryptovirology is the study of the uses of cryptography applied to viral applications. There are two general application scenarios (Anandrao Shivale, 2011). The first is polymorphism: where the virus uses encryption to change and make unreadable part of its code. The second is the crypto viral extortion: where the virus encrypts a target system's files. The attacker then demands a ransom to hand over the key to regain access to the hostage files. The author provides analysis of a Trojan attack that used the crypto viral extortion and further giving counter measures that can be implemented to protect against such attack. The uses of encryption as they are applied by AI algorithms are covered more thoroughly in the next section.

Encryption Through AI Algorithm

Encryption in AI algorithms is a field that is of recent interest, and based on the reviewed literature, had its beginning with the work of (Pointcheval, 1995). The objective of the field is to use the AI or machine learning algorithm to make a message unreadable. A particular example is given in (Kanter et al., 2002), where two two-layered perceptron neural networks are trained together until they synchronize to form a cryptographic system. A recent effort is that of Ismail et al. (2012), where they use a Multi Layer Perceptron to do symmetric encryption. The analogy is drawn by representing the weights as the public key, and the biases of the hidden and output layers as the private keys. The private keys are agreed by the sender and receiver, and act as private keys in a symmetric encryption scheme. The key length is chosen based on the number of neurons in the hidden layer for the hidden bias, and the number of neurons in the output layer used for the output bias. The keys must be numeric. The network is trained using a variant of backpropagation (using momentum). The public key is then distributed, and the output can be transmitted.

Another area of research is selecting the 'key' in image encryption using genetic algorithms is covered in Bhowmik & Acharyya (2011). The authors' objective is to choose the best chromosome by maximizing the fitness value. This will make the chromosome the best 'key', so that when it is applied to the Blowfish algorithm (Schneier, 2007), the correlations among the pixels become minimized. An alternative approach is presented by Mahmood et al. (2013), who worked with genetic algorithms to use selective encryption to provide security to medical images. Selective encryption is the encryption of selected parts of the image or using different algorithms to achieve better performance in processing the images. The system evaluates: the encryption algorithm to be used, key-length, robustness parameter (correlation or pixel change rate), number of regions, and side information (information needed for transmission of the encrypted data). The objective is to minimize the trade-offs incurred between the processing time and the resulting robustness of the encryption.

A novel approach to using neural networks for cryptoanalysis is presented in Alani (2012). Where the neural network used scaled conjugate-gradient, and was utilized using ciphertext as its input and plaintext as its output. The network is assumed that it will retrieve plaintext from cyphertext not used in the training, if the same key is used.

Adversarial Machine Learning

The security of machine learning has attracted recent attention due to the proliferation of machine learning algorithms in security applications, such as spam detection and intrusion detection. The interest lies in that these systems are subjected to an adversary whose purpose is to constantly change behavior to avoid detection and subvert the system. This contrasts with traditional applications of machine learning where the distribution is assumed to be stationary due to the slowly changing nature of the phenomena traditionally studied. In Barreno et al. (2006, 2010) a taxonomy for attack categorization is presented by the authors, in which they specify the influence, security violations, and specificity of attacks as the central axes of the taxonomy. The proposed taxonomy allows categorizations of the attack, as well as exploring the defenses that are available to counter such attacks.

In characterizing the domain of adversarial learning, the attack's effectiveness is dependent on several factors, such as the knowledge of the attacker. A major principle that should be followed when studying adversarial machine learning is Kerckhoffs's principle that states "the security of a system should not rely on unrealistic expectations of secrecy" (Huang et al., 2011). The initial study under this assumption was done by Dalvi et al. (2004), where they used a similar approach to evolutionary game theory. They assumed that adversary can use an optimal plan, and that the Classifier also uses an optimal classification to counter the inputs of the attacker.

In Lowd & Meek (2005), the authors present an alternative to Dalvi et al. statement of the problem and propose the adversarial classifier reverse engineering (ACRE) problem. As opposed to obtaining an optimal strategy, ACRE assumes imperfect knowledge on the part of the attacker, and that the aim of the attacker is to identify the outcomes of queries that provide insight into a classifier's inner workings. The ACRE problem relies not just on the attacker's knowledge of the classifier, but also on the adversarial cost function, which is the cost associated to the usage of particular instances, depending on the complexity of the implementation. In other words, under ACRE, some implementations of the attack might involve less effort to carry out to extract information, than others.

Nelson, Rubinstein, et al. (2011) expand the notion of ACRE by generalizing the problem in two different ways. The first generalization opens the analysis to many classifiers that are not convex and utilized in practice. The second generalization deals with evasion techniques which consist of the attackers ability to refine the method of sampling data to see if better estimates can be obtained. The new attack strategy also contemplates additional feedback from the training distribution, such as the information gleaned by the adversary when he obtains samples from the training data, which may become representative of the underlying distribution. The authors also try to delimit the available information that the attacker has with respect to the mappings to the feature space.

A different approach to Nelson, Rubinstein, et al. is explored in W. Liu & Chawla (2009), where ACRE is used under the assumption that adversarial machine learning is a Stackelberg game. The Stackelberg game model assumes that a leader (the adversary) will make the first move, and the follower is the learning algorithm. The follower reacts to the leader, and it is assumed that the leader will know this and will maximize the payoff, and in turn so will the follower. The authors used genetic algorithms simulating the scenarios of the adversarial Stackelberg game. The simulations always ended with a solution where the Stackelberg equilibrium was effectively found.

A more general and relaxed approach to the analysis of adversarial learning is done by Laskov & Kloft (2009). The authors take a different approach to the other authors in that they provide an analysis procedure consisting of four steps. The first step consists of axiomatizing the learning procedure and the attack process. The second step relies on a detailed specification of the attacker's constraints. The next step consists of specifying the various attack scenarios and identifying the optimal ones. The final step consists of the determination of the bounds for the selected optimal attack scenarios specified in step three. The use of this strategy led the authors to conclude that applying the steps to practical problems led to more optimistic results on the part of designing effective algorithms to counter adversaries.

Nelson, Biggio, & Laskov (2011) analyze the risks of learning within an adversarial setting through a statistical learning framework. However, the exposition is theoretical, as opposed to the preceding discussion of Laskov & Kloft, and cannot be calculated in real case scenarios. Nonetheless, by arriving at bounds for a particular contamination, the model may yield worst case scenarios on which to provide upper bounds on adversarial capacity. These worst case scenarios can then be used as criteria in the selection of algorithms for contamination scenarios. The authors present the analysis starting with the scenario where the contamination exhibits itself as an outlier. The second scenario is the data perturbation where the data points are perturbed by noise. This noise is intended to displace the data points to the attackers' advantage. The next scenario is label flipping where benign and malicious labels are interchanged. Finally, the authors present the feature constrained outlier scenario, in which the features themselves are corrupted. These scenarios are then used by the authors to measure the stability of the classifier subjected to contamination. The derived measure to measure the stability is referred to by the authors as the classification robustness. This measure translates in linear classifiers as the rotation angle of the hyperplane under contamination.

Biggio et al. (2013) use an adversarial model that describes the adversary's goal in terms of a utility loss function. The model also contemplates the adversary's target system and the adversary's capabilities in similar organizational manner as that of Laskov & Kloft (2009). The model is used by Biggio et al. to characterize attacks on systems using gradient descent. The objective is to fool the algorithms, such as neural networks and support vector machines, into classifying samples which have close relationship between them (as in the case of similar handwritten digits). The caveat is that the attacker must guess the underlying distribution, and use data points from densely populated regions to carry out a successful evasion by using the closeness between benign and malicious data. The framework was also used in defeating biometric systems Biggio et al. (2012), as a poisoning attack that gradually compromises the biometric template used to store the biometric signature until the benign template is replaced by the malicious one.

Overall the field of adversarial learning can be compared to an arms race (Roli, 2013), where the attacker first analyzes the machine learning algorithm and tries to use it to his advantage; then the designer of the machine learning algorithm system counteracts with an update to the system. The process then repeats itself in a recurring cycle.

Neural Network Trojan

Implementation vulnerabilities for neural networks were described in Geigel (2013), nonetheless, these vulnerabilities can be generalized to other learning algorithms and can be classified into three categories. The first category are vulnerabilities that arise from the data, and aim to poison the data (Kloft & Laskov, 2007; Nelson & Joseph, 2006), by adding specifically crafted data to influence the learning. The specific taxonomy in this category of attacks is given in Barreno et al. (2010). The second category of vulnerabilities takes advantage of the algorithm behavior by compromising the integrity of the algorithm, encompassing attacks where the attacker uses the training cycle to embed malicious payloads in order to create a compromise. The third category is a compromise on the programming or the hardware where the learning algorithm runs. While the three vulnerabilities can be exploited independently, the use of the three categories to launch an attack can fall under the category of Trojan horses (Thimbleby et al., 1998).

The implementation of the NnT assumes that the elements of Payload P' can be modeled as independent from one another. This assumption is needed, since insertion of P' into a benign training data B' may lead to some elements of P' to be categorized as B', as well as some elements of B' into P'. The relationship between any vector $x \epsilon X$ on the mixed data set that contains elements of the category C is expressed in terms of the conditional probability density:

$$\phi_i\left(X|C_\gamma\right) = \int P\left(x|C_\gamma\right)\, dx \tag{1}$$

Where the category label γ may be either B' or P'. Since the training set mixes both B' or P', this gives rise to the notion of a mixture of densities expressed as:

$$\sum_{i=1}^{2} \phi_i\left(X|C_\gamma\right) = \phi_1((X|C_{P'})) + \phi_2((X|C_{B'})) \tag{2}$$

This last formula gives rise to the possibility of N mixtures of payloads that can be expressed in terms of a more general formula:

$$\Omega = \sum_{i=1}^{n} \phi_i \left(X | C_\gamma \right) \tag{3}$$

This generalized mixture of conditional densities leads to the problem of creating a Trojan for a neural network as the embedding problem that can be postulated as:

> *"Given a Neural network N, can we use a training data based on the mixture of conditional probability densities Ω to train the neural network N to memorize the known distribution payload P' that is embedded as part of Ω, and generalize the distribution of the benign data B'?" (Geigel, 2013)*

The embedding problem for the neural Trojan is based upon several underlying assumptions listed below:

1. The attacker controls the process of development of the algorithm implementation and the end user does not know that the application has an unwanted side effect.
2. The attacker controls the training and testing process of the Neural Trojan implementation.
3. The attacker can construct the distribution of the payload P' in an arbitrary fashion.
4. The attacker can manipulate the parameters of the algorithm.
5. The attacker can manipulate encoding of the data to his advantage.
6. Any lack of convergence can be compensated at the post processing stage.

These assumptions allow the attacker to deploy the Trojan under reasonable real world scenarios. The NnT is implemented on a basic backpropagation neural network and on the Neural Network Trojan with Sequence Processing Connections (NNTSPC). In the NNTSPC, the backpropagation neural network is followed by a gating function that is activated by one of the inputs. The Trojanized networks are tested on a synthetic data set and on the Congressional Voting Records Data Set (Asuncion & Newman, n.d.). Due to the different training parameters, the NnT can encode the payload under different weight matrices. This ability to achieve a high mutation rate of matrices gives the NnT a high polymorphic capability, making it extremely difficult to detect in the encoded phase, and it is only after the Trojan is decoded and executed that it may be possible to discern its compromised nature.

Summary

The field of machine learning is being actively studied as the trend of using machine learning in diverse areas continues to grow. As the applications for machine learning in security related areas continue to grow, so do the threats to the algorithms that make up these systems. Current research shows that the need to adopt Kerckhoffs's principle is essential in providing the necessary robustness to withstand attacks on machine learning algorithms. The research has provided different assumptions and criteria on which to judge Kerckhoffs's principle based on theoretical upper bounds, as well as reasonable practical assumptions. As the field matures, it is expected that several of the related fields such as cryptoviruses, AI encryption, and Trojans see a converging path. Recently, the convergence has only been slightly touched by the implementation of the Neural Network Trojan.

Chapter 3

Methodology

Overview of Research Methodology

The proposed approach starts with the analysis of the Unsupervised Learning Trojan from the basic assumptions of the NnT, and then shows how the unsupervised Trojan deviates from the supervised learning assumptions. The discussion then describes the analysis of the unsupervised learning parameters that need to be taken into account to carry out a successful attack. Three algorithms, four data sets, and eight payloads were arbitrarily selected as representatives of the unsupervised learning field. An individual background is given on each of the individual algorithms, and based on the general characteristics of the algorithms, the procedures for payload encoding were adjusted accordingly. An experimental setup is devised to simulate the steps required by an adversary before the attack. These steps can be taken to be the "training" set of the experimental procedure. An evaluation criteria is given based on the measurements obtained during the "training session" which serve as selection criteria of the best parameters to use in the actual compromise. The actual compromise would represent the "testing" procedure of the experimental setup.

The Unsupervised Learning Trojan Density Problem

For the Unsupervised Learning Trojan, the assumptions given for the NnT embedding problem described above must be reevaluated, as they either would need to be modified or would not apply. For the NnT, these assumptions allow the attacker to have an ideal scenario, where the attacker can manipulate both the training data set and the payload without any restriction, so that he can arbitrarily construct any training distribution. In such cases, the attacker has perfect information and is able to choose the strategy that maximizes his utility. This means that attacker can successfully manipulate the training so that the prior probability of the compromise can be theoretically chosen to be unity. Under this scenario, the attacker can train the neural network to a desired degree of accuracy vs. the polymorphic ability of the attack. This leaves the attacker with just needing to set the appropriate mixture Ω to achieve his goal.

This is not the case for the Unsupervised Learning Trojan. Under this new type of attack, the attacker does not have control of a training set, and now the parameters need to be estimated instead of being available for manipulation. This process takes the form of density estimation of the distributions. Since the benign data and the payload are given to the algorithm without labels, the goal of the compromised algorithm is to discriminate between the benign data and the payload. The question to be answered is whether an unsupervised algorithm is capable of discriminating between unlabeled malicious samples and the benign data. We can start the analysis with Equation 3, which now becomes:

$$\Omega = \sum_{i=1}^{n} \phi_i \left(X|\psi\right) \tag{4}$$

where ψ are the parameter vectors for the class. In addition, the attacker no longer has the full control of the prior probabilities as in the NnT, and the mixtures are now determined by:

$$\Omega = \sum_{i=1}^{n} \phi_i \left(X|\psi\right) P\left(\psi\right) \tag{5}$$

The question that immediately follows is: If given a Unsupervised Learning Trojan, can the attacker arbitrarily manipulate the cluster selection?

The goal of the attacker will be: To achieve a mixture, given two densities ϕ and ϕ_{i+1}, such that:

$$\phi\left(x|\psi_{comp}\right) \neq \phi_{i+1}\left(x|\psi_{Ci+1}\right) \tag{6}$$

where ψ_{comp} is the composite parameter vector for the attackers mixture. This argument can then be generalized to any number of mixtures.

The equations above must satisfy the following relation for the parameter vectors ψ_{comp} for the class:

$$\psi_{comp} \neq \psi_{Ci+1} \tag{7}$$

and must also meet the requirement of the Trojan elements $x_{1i}^{T'} \in X^{T'}$, where $X^{T'}$ is the vector class of *Trojanized inputs.* To meet these requirements, the attacker must estimate the volume density spread of the Trojan cluster and maximize the inter cluster distance with the benign data to avoid overlaps between the payload and the benign information that may hamper the execution of the compromise. It can also be stated that: in absence of any information about the Trojanized inputs, the assumption will be made that the Trojan elements will be treated as random vectors, whose behavior can be described by a parametrized form, and will behave the same along all dimension of the vector space on which it operates. This will simplify the analysis of the Trojan behavior and the benign data, and will also have a big impact on the encoding of the Trojan payload P'.

The volume density of the Trojan attack may be determined by:

$$P\left(Trojan\right) = \int\limits_{V} p\left(x\right) dx \tag{8}$$

While the attacker does not have a training set to compromise, as in the case of the NnT, it is assumed that the attacker has access to the algorithm, its implementation, and to information regarding the goal of the implementation. By having access to the goal of the implementation, he can sample elements from the population and draw inferences as to the possible clusters that might be formed during an actual implementation.

To model the compromise, the sampling done by the attacker will have a sample N that has embedded a set of compromised points m_{Trojan}, which are the result of the inputs $X^{T'}$ into a solution space, acting as the payload P'. The probability $P\left(Trojan\right)$ of m_{Trojan} falling on the correct cluster defined by a density function $p\left(x\right)$ covering a volume v that lies on an interval V is given by:

$$P\left(Trojan\right) = \int\limits_V p\left(x\right)dx = p\left(x\right)v \approx \frac{m_{Trojan}}{N} = M \tag{9}$$

$$p\left(x\right) = \frac{M}{v} \tag{10}$$

The attacker must further assume that there will still be scattering of the Trojan elements around the centroid. As long as the attacker has no specific information on the Trojan density behavior, the $p(x)$ can be modeled in general terms by:

$$p(x) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{\left(2\pi\sigma^2\right)^{1/2}}exp\left(\frac{-\left(x-\mu\right)^2}{2\sigma}\right) \tag{11}$$

which gives the volume v in terms of the σ, and $\frac{m_{Trojan}}{N}$ as μ.

A similar definition can be done for the benign data that the attacker will try to keep in a separate cluster but cannot avoid scattering. As the attacker defines the specific Trojan and gathers information on the data set, he may acquire certain information on the centrality of the points and the scattering, which may or may not be homogeneous. Therefore, a more general description becomes:

$$\Omega' = \sum_{i=1}^{n}\phi_i\left(X|\mu,\sum\right) \tag{12}$$

where Ω' contains ϕ_i densities that belong to Payload P', and the component density $\phi_i\left(X|\mu,\sum\right)$ is a D dimensional Gaussian (Murphy, 2012), where $\sum_i$ is the covariance matrix D and $Transp$ is the transpose:

$$\phi_i\left(X|\mu,\sum\right) = \frac{1}{\left(2\pi\right)^{D/2}\left(\|\sum_i\|\right)^{1/2}}exp\left(-\frac{1}{2}\left(x-\mu\right)^{Transp}\sum\nolimits_i^{-1}\left(x-\mu\right)\right) \tag{13}$$

The next section will lay the groundwork for the actual attack deployment, followed by the analysis of a practical example, taking under consideration the model explained above.

Attacker's Analysis of the Unsupervised Learning Trojan Density Problem

The compromise scenarios presented are based on the assumption that the attacker can have access to the samples and approximate the behavior of the attack with the obtained samples. The analysis will focus on testing the assumptions before launching the attack on a compromised algorithm and obtain the equivalent of cluster validity measures (Davies & Bouldin, 1979; Bezdek & Pal, 1998; Kim & Ramakrishna, 2005; Deborah et al., 2010). This analysis carried out by the attacker occurs during a calibration phase of the attack, which is similar to the validation phase under supervised learning, where the attacker, using the compromised algorithm, will tweak the parameters of the attack before the actual compromise takes place.

While the attacker does not have full control of the prior probability for all the data, the attacker may impose some constraints on the payload's prior $P(\psi)$ based on strategic selection of the payload's characteristics and encoding. If no other information is obtained during the calibration phase of the attack besides the sampled obtained data, he must assume a certain distribution of the unknown example and this will be based on equation 13. Under special scenarios where the attacker chooses to attack certain selected data sets, the attacker can make further assumptions under most real scenarios of the spread of the densities of the benign data. For example, given a census data, the attacker can infer that the probability that a sample X from the data is over 100 years of age and has Y other characteristics. While the aforementioned analysis may be used it is a strong assumption that will not further be pursued in this study and the basic starting point of equation 13 will be used.

The attacker can leverage the intercepted data to select the appropriate subspace for the Trojan, and from there, he can approximate:

$$f(N, m_{Trojan}) \approx f_{training}(M) \tag{14}$$

where $f_{training}$ is the training derived function using sampled data. This last equation justifies that an analysis of the calibration phase, under the reasonable assumptions and properly executed attack components, the attacker can, in principle, approximate the underlying distribution that the algorithm will use in the compromise. This reasoning, in turn, allows for the attacker's analysis of the clusters under a calibration or "training" phase.

The starting measurement to be used is an approximation to the equation 10, using the obtained data set during the calibration stage of the attack. The approximation in D dimensions is the cluster density DY that can be obtained by:

$$DY_{Trojan} = \frac{M_{trojan}}{\sqrt{(Max - Min)^2_{D_1 Trojan} + ... + (Max - Min)^2_{D_n Trojan}}}, \tag{15}$$

where M is the average number of sample points in the Trojan cluster over the total number of elements of the Trojan, and the denominator volume measurement can be made on n dimensions D of the Trojan cluster. The density for the benign clusters can be defined similarly.

The use of equation 15 can be adversely be affected by the misclassification of samples and malicious data during the calibration phase. To quantify the possible misclassification due to the overlap in the tails of the assumed Gaussians, the attacker can use the degree of similarity DS between the clusters. The DS measure is intended to determine the degree of overlap between the clusters. The formula for the degree of similarity DS is given below:

$$DS = \frac{NMC_{Trojan} + NMC_{Benign}}{N} \tag{16}$$

where N represents the number of points in the two clusters, NMC_{Trojan} is the number of Trojan elements that were misclassified as part of the benign cluster, and NMC_{Benign} is defined similarly. The goal is to minimize the number of misclassified elements on both cluster elements. The formula is also applicable to more than two clusters, since the misclassification is centered on Trojan elements and inclusion of other members into the Trojan cluster.

The density calculation 15 is very useful in providing density information, however within the enclosed volume, the distribution of samples may be skewed. A measure of “skewness” will be calculated by using Degree of Cluster Distance (DCD) given by:

$$DCD = \frac{\sqrt{(CL_{Trojan} - CL_{Benign})^2_{D_1} + \ldots + (CL_{Trojan} - CL_{Benign})^2_{D_n}}}{\sqrt{(Max - Min)^2_{D_1} + \ldots + (Max - Min)^2_{D_n}}}, \tag{17}$$

where CL is the specific centroid location coordinates, and $(Max - Min)^2$ is the squared distance of the maximum and minimum of the data points along the dimension D of the clusters. The DCD grows as the distance between centroids increases (the numerator) or as the spread distance between the dimension D decreases (the denominator). The Trojan designer will try to maximize the DCD.

If the DCD is too small, then the cluster should be reassessed or eliminated due to the closeness of the clusters. What this implies is that cluster centroids that are too close with little spread among the data might be too similar for discriminative assignments.

The attacker can use the information obtained to establish the measures given above and make reasonable estimates of the attacks effectiveness. An important question to answer is whether he can further take advantage of the Gaussian assumption to simulate the rest of the data that the user will work with during the compromise. This will lead to two different scenarios to test, one where the attacker uses only the intercepted data, and the other where the attacker extends the obtained data using the parametric assumptions to further increase his chances of success.

The Cluster Identification Sub Problem

The cluster identification sub problem is the distinguishing of the malicious cluster from those identified by the algorithm as benign. This presents several challenges, since the algorithm’s discriminative measures are not designed to discriminate clusters, just separate them. Cluster discriminative measures can be altered to identify one or more compromise clusters using either intrinsic or extrinsic attributes of the malicious payload.

Intrinsic properties are those properties that can be found by just analyzing the properties within a single cluster. Some of the properties that can be used are:

- Size of the sampled data
- Location
- Uniformity
- Density
- Shape
- Surface area
- Topology
- Inter point relationship
- Temporal behavior

The size of the sampled data within a specified cluster can be used as identification when it reaches a certain threshold, where it acts as a trigger for the payload to be decoded, executed or both. For example, if the amount of data in a particular cluster exceeds the normal count of n samples, then this serves as the trigger for the next data elements that form part of the cluster to be malicious. The strategy can also be implemented by using the location of the cluster or when the cluster moves its position to a certain predetermined location. The location trigger can be obtained if, for example, the location of the cluster falls within the lower right quadrant of the usual locations prescribed in a Cartesian coordinate system, tagging this cluster as the malicious one.

The clusters uniformity, density, shape, and surface area can also be used to trigger the cluster identification in a manner similar to previous attributes. When a particular identified cluster reaches the predetermined uniformity measure or density, then the cluster is identified as malicious.

Using topology as an intrinsic property that identifies the malicious cluster is more difficult to implement. However, the topological property of connectedness in a region can be implemented by partitioning the clusters into equal size subspaces. If a subspace within the cluster is empty, then we can say that two regions separated by this subspace are not connected and may lead to identification of the cluster as malicious.

Identifying malicious clusters by using inter point relationship within a cluster can be carried out in one of several ways. An example would be to measure the angle of each point relative to a single identified point in the cluster, such as the cluster's centroid. Other interrelation attributes may be the Euclidean distance relative to the cluster's centroid, positioned in a specific order, such as clockwise or counterclockwise measurement execution.

The temporal behavior can be broken down into three sub properties. The first strategy is to take advantage of the frequency of occurrence in a specific temporal interval in order to activate the payload. The second strategy is to use a specific input sequence, in an order for which its probability of occurrence is highly improbable. In the third variant, the sequence elements are not relevant, but the timing of arrival into an online algorithm will be the activation trigger.

Extrinsic properties are those that set the malicious payload apart from the rest of the benign data, by comparing the behavior of the malicious cluster(s) from those that are benign. The same properties used under the intrinsic analysis can be used as extrinsic properties. To give several examples, the size of the sampled data can now be measured as the difference between clusters, as opposed to just a predetermined cluster, (e.g. make the cluster an outlier that can be distinguished as the attack payload). The location property can be measured as the location relative to other clusters that are being analyzed. The same extension used in these two examples can be done for the other properties as well.

A final example would be the density relationship of the clusters, where the comparison between clusters can be represented as an imbalanced set. Under this scenario, one of the classes is underrepresented with respect to other classes that are present in the data set, making the relative density of the cluster less than the others. The attacker can take advantage of this imbalance to determine which of the clusters is the Trojan payload and activate it.

Payload Activation and Decoding

The payload activation and encoding strategy will depend on the specific data set qualities that the attacker can use to his advantage. The strategy can be divided into two separate stages: payload activation and encoding. The activation and decoding of the payload are directly related to the cluster identification sub problem, since the identification leads to both the activation and decoding of the payload. The activation and decoding of the payload may occur at the same time or at different stages of the exploit.

In the activation phase, the compromised algorithm detects that one or more data items are part of the payload of the attack. While offline algorithms may not have a specific sequence of data input or are using randomized ordering for the input data, it is necessary to have an ordering for payload decoding.

The payload decoding is described by:

$$\begin{aligned} P(S_{success}) &= P\left(s_1, \ldots, s_n\right) = \prod P\left(s_j | s_{j-1}, \ldots, s\right) \\ &= P\left(s_j | s_{j-1}, \ldots, s\right) P\left(s_{j-1} | s_{j-2}, \ldots, s\right) \ldots P\left(s_2 | s_1\right) P\left(s_1\right) \end{aligned} \tag{18}$$

The success of carrying out the probability constraint in Equation 18 lies in the ability to have a successful activation and subsequent strategy using intrinsic or extrinsic properties that can take advantage of the *well ordering principle.* That is, in this work, a well ordered sequence using this principle is a decoding sequence using an intrinsic or extrinsic property that can be put in a one to one relationship with the natural numbers, such that the first element of the sequence is the first element, and the second element of the sequence is the second element, and so on. This one to one correspondence can in turn be translated into a temporal sequence using this one to one relationship.

Practical Considerations

The implementation phase deals with inserting compromised elements into the algorithm, such that when it receives the desired data inputs, it will carry out the desired behavior of the attacker. The generic architecture of a typical unsupervised learning algorithm is given in Figure 1, which presents the architecture in three parts: the preprocessing part, the main algorithm, and the post processing part of the algorithm. The proposed attack focuses on compromising the algorithm and using the post processing part of the architecture to decode the payload in an optimal manner. While it can be advantageous to prefilter the data in the preprocessing part to further guarantee the chances of success, it will not be covered in this work.

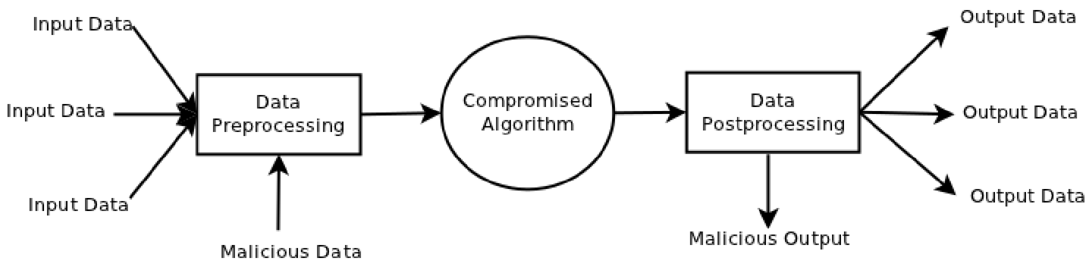


Figure 1. Diagram of a compromised algorithm

The generic steps to carry out the compromise are given in the Algorithm 1 shown below:

Algorithm 1 Generic steps carried out by an attacker to construct an Unsupervised Learning Trojan

```
1. Determine which intrinsic or extrinsic measurements to add
   as part of the compromised code and the decoding sequence strategy.
2. Carry out the analysis of the Unsupervised Learning Trojan
   density problem. If successful, go to step 4, else continue to step 3.
3. Determine stabilization techniques that are necessary, and return to step 1.
4. Substitute the benign code with the compromised algorithm.
5. Inject the payload in the same manner as steps 1 through 3, to achieve the predetermined
   objective.
```

The first step is to determine which intrinsic or extrinsic property will be used to trigger the payload activation and decoding of the attack. The selection does not exclude the possibility of combining multiple properties to carry out the compromise. An example would be using the location of the cluster within a predetermined location as the activation trigger, and then use the inter point distance to decode the payload. From this step, the attack can then be tested by inserting the payload into the benign data.

The second step involves the analysis of Equations 15 through 17 to determine the attack's efficacy. A careful tradeoff must be considered by the attacker at this stage between making a successful attack, and at the same time minimizing the possibility of detection due to a recognizable pattern that characterizes the payload.

The third step consists of stabilizing of the cluster, if required. The stabilization is aimed at manipulating the clusters of benign data to maximize the probability that the compromise is successful. A description of cluster stabilization can be carried out using the influence and security violations axis (Barreno et al., 2010). The influence axis allows for causative and exploratory attacks, and deals with the adversary's ability to influence or observe the training data. The security violations axis describes the type of security violations that the attacker can make, and consists of integrity and availability attacks. Integrity attacks refer to allowing false negatives, while availability attacks create denials of service.

In this work, the influence axis will be modified in the form of malicious data influencing benign data, so that the attacker can attempt to maximize the inter cluster distance. In turn, the security violations axis will be used as access denial, so the attacker can force the clusters into a stationary state, where the attacker can inject the payload without being overtaken by the benign data. This deviates from Nelson & Joseph (2006) approach, where the objective is to subvert a detector by shifting the non-anomalous region hypersphere into the space where the attack vector is located, so that the detector would treat it as non-anomalous.

If the compromise is not successful at steps 2 and 3, the attacker may carry out further cluster stabilization techniques such as: denial of service, inserting more data to balance the cluster to the desired outcome, or return to step 1 to redesign the attack. If the attack is successful, the attacker can then proceed to step 4, which is to substitute the benign code with the Trojanized code and let it stay dormant until step 5 can be executed. Step 5 will be the actual injection of the payload into the benign stream, which will depend on whether the algorithm is online or batch. The attack will mimic the simulation of data injection carried out by the attacker in steps 1 through 3.

Algorithms and Data Sets

This section shows the necessary setup to carry out the Unsupervised Learning Trojan with three selected unsupervised algorithms to demonstrate the analysis and concepts that the attacker must take into account for a successful attack. The three selected algorithms are:

- K-means algorithm (MacQueen, 1967)
- Kohonen Neural Network (Kohonen, 1982, 1989)
- Adaptive Resonance Theory (ART-1) (Carpenter & Grossberg, 1998)

Within this work, implementation of each algorithm were be based on the original algorithm to increase the vector surface of attack, since it is more probable that the original algorithms have a wider adoption than more recent variants of the algorithms. It is also easier to generalize from the original algorithm to the more recent variants, than the other way around. Each of the three algorithms presented a different aspect of unsupervised learning and different challenges to the deployment of the ULT. While K-means measures the centroids on a real dimensional space, Adaptive Resonance is based on binary input variables. Adaptive Resonance also has a growing number of clusters, while Kohonen and K-means have fixed number of clusters. These differences will be covered during the analysis of the compromise.

The selected data sets to be analyzed were taken from the UCI Machine Learning Repository (Asuncion & Newman, n.d.). The four selected data sets are shown in Table 1. The choice of data set for a particular algorithm will be made based on the algorithm's ability to process a specific attribute type and the need for multiple data sets per algorithm.

The vector entries within each of these data sets will be divided into "training", "validation", and "testing" sets. The "training" set will be the data available to the attacker, and the "testing" will simulate the algorithm in the attack environment. The sets were chosen to reflect possible power, medical, and scientific analysis, to cover some possible real world scenarios under which the compromise might be possible. Also, the data sets were selected with both integer and real values, to simulate various types of possible encoding strategies. Finally, the data sets were also selected varying from "low" to "high" instance availability, to show various loose, as well as constrained access, to data scenarios.

Data set	Attribute Type	Instances	Attributes
KEGG Metabolic Relation Network	Integer, Real	53,414	24
Individual Household Electric Power Consumption	Real	2,075,259	9
Statlog Project (Shuttle)	Integer	58,000	9
Water Treatment Plant	Integer, Real	527	38

Table 1: Proposed data sets

Analysis Carried Out by the Attacker on the Data Sets and Malicious Payload

The analysis to be carried out by the attacker will be broken down into several stages. The first stage is the analysis of the Trojan payload as in Geigel (2013). The Trojan Payloads were selected from the Exploit Database (http://www.exploit-db.com) to serve as payload. These are given in Table 2

The main focus in the ULT payload is the frequency of occurrence of binary elements, and that the encoding surface be as small as possible so it does not overlap with benign clusters. On some algorithms such as K-means the size of the payload will help in moving the cluster belonging to the malicious code further. On the other hand, the K-means will need specific sizes to be processed on an N*N Kohonen layer since size could interfere with the success of the compromise.

Once the payload is encoded appropriately, the analysis of modifications of the algorithms to Trojanize them will be carried out. The next stage of the analysis will be the process of injection of the payload into the benign data. This stage will concentrate on analyzing the trajectories of the clusters and the measures described in the section of the attackers' analysis of the clusters.

	Exploit	Size (in bytes)
1	Linux x86 ASLR deactivation	83
2	Linux_x86 Polymorphic ShellCode - setuid(0)+setgid(0)+add user 'iph' without password to _etc_passwd	124
3	OSX_Intel reverse_tcp shell x86_64	131
4	OSX/Intel - setuid shell x86_64	51
5	win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode	112
6	win32/xp pro sp3 (EN) 32-bit - add new local administrator	113
7	Windows XP PRO SP3 - Full ROP calc shellcode	428
8	Windows XP SP3 English MessageBoxA Shellcode	87

Table 2: Proposed payloads

Compromising K-means Clustering Algorithms

This section introduces the compromise of the K-means clustering algorithms by an ULT instance. A brief review of the algorithm is given to describe its main functionality. The discussion is followed by the attack strategy to be implemented on the algorithm and the data.

K-means clustering was first proposed by MacQueen (1967), and since then, it has become one of the most important clustering algorithms in machine learning and data mining fields. Its success can be attributed to its simplicity, while at the same time being very useful for the purpose of determining the centroid on a predetermined number of clusters. The K-means algorithm processing can be broken into the following stages:

1. Randomly initialize the centroids.
2. Assign the sample points according to: $|x - \mu_j| < |x - \mu_k|$.

3. Calculate the new centroids using the assigned centroid and the samples closest to it using $\mu_k = \frac{1}{N_k}\Sigma_{q=1}^{N_k} x_q$, where q are the elements assigned to a cluster k.

4. Update cluster center based on step 3.

5. Return step 2 until convergence.

The vulnerability analysis carried out by an opponent is based on Algorithm 1 and customization to the K-means clustering, as in Algorithm 2. The selection of cluster separation as the trigger is based on the sample point assignment (step 2) of the K-means algorithm. As the separation between malicious payload and the benign clusters increases, so will the probability of attack's success.

Algorithm 2 K-means Trojan compromise procedure

```
1. Use the centroid location as a trigger for the K-means Trojan. The decoding will use the
   angle of the payload elements to determine the membership of elements as part of the payload
   $P'$.The decoding strategy will read the points in a counter clockwise manner relative to the
   centroid and decode the values as the angle relative to the previous point in the cluster.
2. Identify which cluster will be used as the malicious one, and quantify the error.
3. If necessary, unbalance the data set to maximize the inter cluster distance and continue to
   step 4. Else, return to step 1.
4. Compromise the target by substituting the benign code with the compromised algorithm.
5. Inject the payload to achieve the predetermined objective.
```

The attack's complexity lies in that the attacker must guess the predetermined number of clusters. Therefore, the attacker must carry out his tests under different cluster configurations that he predicts the user may use. The attack may also need to be supplemented by the use other techniques such as data poisoning, so that the data set becomes imbalanced. The cycle will be repeated under different configurations until the attack becomes viable. The decoding strategy will take the centroid of the compromised cluster as point of reference on which to start a clockwise decoding sequence based on the relative angle of each previous point in the sequence.

Payload Encoder and Decoder for K-means Compromise

Out of the intrinsic and extrinsic properties of the malicious clusters, the angle between data points was chosen. The particular angle encoding can then be expressed in terms of polar coordinates, specified by an angle θ and a ray r. The selection of the angle is based on the hexadecimal code chosen as payload of the attack. Each malicious data point angle is encoded using two hexadecimal numbers. This gives 256 possible angles for the malicious data points. Once the angle is converted from hex to decimal representation, the length of the ray is arbitrarily given by the adversary, and will remain fixed throughout the encoding. From these two pieces of data, the attacker can calculate the new *(x, y)* coordinates by using the following formulas:

$$x = r * cos\,(\theta) \tag{19}$$

$$y = r * sin\,(\theta) \tag{20}$$

A symbolic representation of this process is shown in Figure 2 (a). For the attacker, the next step is to place the initial payload points in an offset, as shown in Figure 2 (b). The offset will be chosen according to the feature landscape targeted for compromise.

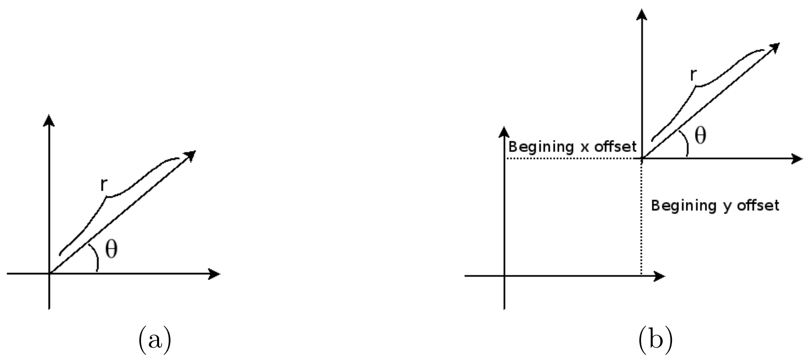


Figure 2. Encoding of: a) the malicious angle and b) offset to the starting coordinates specified by the attacker

An initial representation of the malicious points for the Linux x86 ASLR deactivation Payload is shown in Figure 3. While this representation appears plausible as an attack payload, it has a particular drawback, since the decoding sequence cannot be guaranteed for all payload variations to be unique. Although the occurrence of such error is small, an encoding scheme with possible zero error in encoding is preferred.

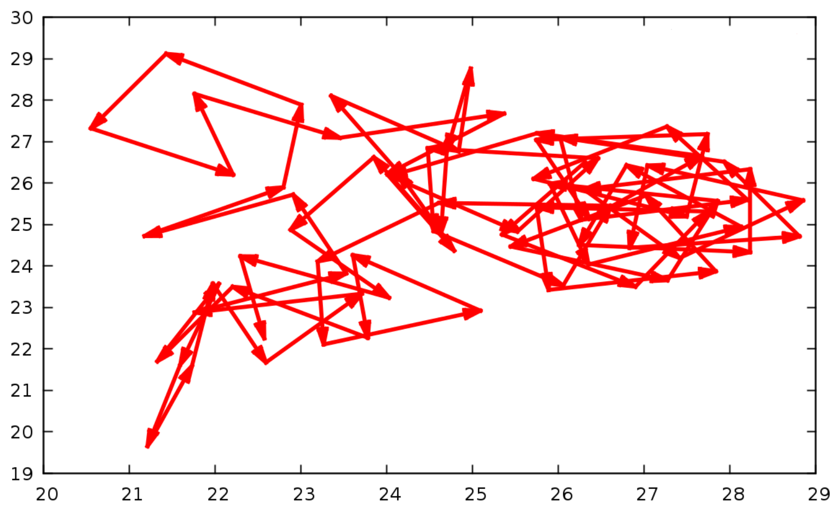


Figure 3. Representation of malicious points for Linux x86 ASLR deactivation payload

A second alternative is to rearrange the attack points to form a hull, and then center the random points around the compromising hull. This procedure has the advantage of including additional points to the attack vector that are used as padding for the encoding, and can further help in attracting the cluster into the required range.

The method of constructing the compromising hull is to calculate each attack point using an encoded angle at a distance r, as shown in Figure 4. The angle of the padding will be calculated using the following formula:

$$\theta_{padding} = 360 - \theta + \Delta \tag{21}$$

where θ is the angle to be padded and Δ is the compensating angle. For example, this angle can be 360, divided by the total number of the payload's hex characters. For the

Linux x86 ASLR deactivation Payload, $\Delta = 4.337$.

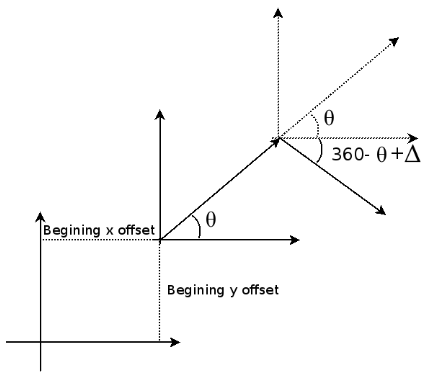


Figure 4. Malicious point encoded with offset and padding angle

The final result of the process described above is shown in Figure 5. The drawback with this simple implementation is that the shape can be easily discriminated from a benign cluster of points. A more complex extension to this algorithm is to use cubic splines (Press et al., 1992), to make more complex hulls which also define the Δ of each padding. For the cubic spline alternative, r is left fixed for the purposes of recognition by the decoder algorithm.

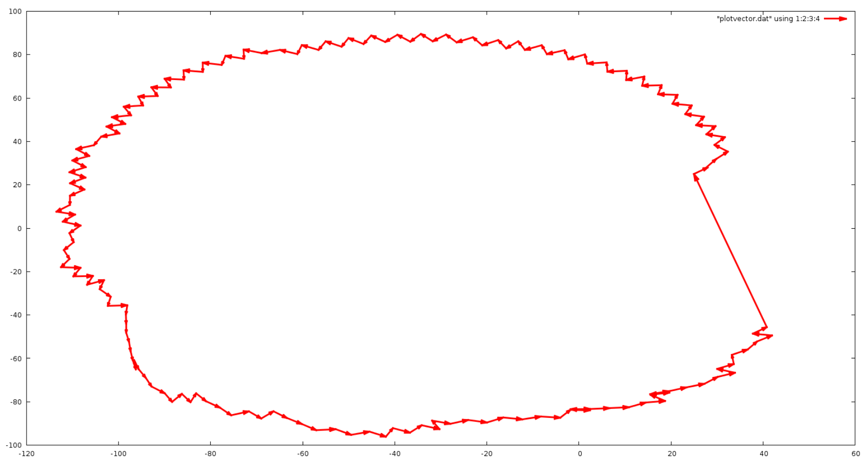


Figure 5. Result of hull creation for Linux x86 ASLR deactivation payload with padding points

The pseudo code for an encoder using 360 degrees as parameter for the hull shape is given in Algorithm 3 . The code is a single procedure that takes 4 arguments: the payload translated from hexadecimal to decimal numbers, the initial x and y positions on which to start the compromising hull, and the fixed ray length. Lines 5 through 10 generate the x and y coordinates with the initial offset, and store them as previous values for the main loop. Line 11 is the main loop. Lines 13-16 compensate for angles that are more than 180 degrees. This compensation makes it possible to achieve the surface without sub surfaces, as in Figure 3. Lines 18-23 generate the x and y points of the payload within the load. The points are inserted into the array at line 24, and the padding angle with its corresponding points is generated in lines 25 through 31, and stored in line 32. The for loop is continued until all the points have been processed, and then the procedure ends.

Algorithm 3 Payload encoder for K-means compromise

```
1  PROCEDURE Generate_hull_of_attack_points(angle list in decimal form,
2                                          initial_x,initial_y, r)
3    average_angle=360/total number of elements in angle list;
4    CALL Insert_into_array (x,y)
5      x= r *cos (phi_tot*PI/180)
6      x_vector=x+ initial_x
7      previous_xpoint=x_vector
8      y= r *sin (phi_tot*PI/180)
9      y_vector=y+initial_xy
10     previous_ypoint=y_vector
11     FOR each of the angles in decimal form list
12         phi=current angle of decimal list
13            IF ( $phi>=180){
14                 CALL Insert_into_array(previous_xpoint, previous_ypoint)
15                 phi=phi-180
16            END IF
17     phi_total=phi+phi_total
18     x= r*cos (phi_tot * PI/180)
19     x_vector=x+previous_xpoint
20     previous_xpoint=x_vector
21     y= r*sin (phi_tot * PI/180)
22     y_vector=y+previous_ypoint
23     previous_ypoint=y_vector
24     CALL Insert_into_array(x__vector,y_vector)
25     phi_total=-phi+average_angle+phi_total
26     x= r*cos (phi_total * PI/180)
27     x_vector=x+previous_xpoint
28     previous_xpoint=x_vector
29     y= r*sin(phi_total * PI/180)
30     y_tmp_vector=y+previous_ypoint
31     previous_ypoint=y_vector
32     CALL insert into array (x_vector,y_vector)
33   END FOR
34 END PROCEDURE
```

The particular decoder procedure initially takes the centroid location as a trigger for the K-means Trojan. Another alternative is to use distance between 2 clusters as a trigger for the K-means Trojan. However, if there are more than two clusters, then the measure should be taken against the malicious cluster and each of the benign clusters individually. If a distance reaches a threshold bigger than N, it will trigger the decoder sequence, as shown in *PROCEDURE Decoder phase1* of Algorithm 4. Once the decoder is activated it will call the *PROCEDURE Decoder phase*2. This part of the code will keep executing the while loop of line 12 until the exploit chain that holds the exploit is completed and convert the candidates to hexadecimal. To process the chain within the while loop of line 12 calls the procedure PROCEDURE f i n d _ n e x t _ p o i n t. This procedure will go through all the points within the cluster boundary assigned to the centroid to determine if they are part of the exploit.

To find if the point is part of the chain two crucial elements must be specified which are: the initial x-y coordinates and the distance r between exploit points. The distance between elements is calculated in lines 21 and 22 of Algorithm 4. The candidate distance is calculated using the euclidean distance in line 25. If the result form the calculations is odd then it is saved into the list of candidates. The even candidates that are eliminated from the list are the ones that represent the padding angles. While these points are not relevant to the payloads content they are part of the decoding mechanism as is shown in lines 23 and 24 where the coordinates are saved to complete the decoding process.

To get the final hexadecimal numbers the Algorithm 4 uses the inverse of the tangent to extract the decimal representation of the hexadecimal code in lines 35 and 37. In line 37, if phi is greater than -400 then 180 degrees are added to undo the addition of 180 degrees of line 15 of Algorithm 3. Additional routines such as opening files and saving files must be included as part of the decoder and must be contemplated in the overall size of the code base that is part of the exploit. Leveraging native routines to minimize the size of the exploit coding “surface” is essential to reduce the possibility of detection on the part of the adversary.

Algorithm 4 Payload decoder for K-means compromise

```
PROCEDURE Decoder phase1(cluster_distance,N, r,initial x, initial y,
                                average angle,list_of_points_from_cluster)
  IF cluster_distance > N
        CALL Decoder phase2(r,initial x, initial y, average angle,
                                list_of_points_from_cluster)
  END IF
END PROCEDURE

PROCEDURE Decoder phase2(r,initial x, initial y, average angle,
                                list_of_points_from_cluster)

  WHILE exploit_chain not completed
        phi = CALL find_next_Point(list_of_points_from_cluster,r,next_point)
        hex_number = CALL convert_to_hex(phi)
        CALL store_value(hex_number)
  END WHILE
END PROCEDURE

PROCEDURE find_next_point(list_of_points_from_cluster,r)
        FOR i=1 to number of points from cluster list
                x= x coordinate from cluster point-previous coordinate_x
                y= y coordinate from cluster point-previous coordinate_y
                previous coordinate_x=x
                previous coordinate_y=y
                length_candidate_r=SQRT(x^2+y^2)
                IF (length_candidate_r = r){
                 IF(i=odd)
                 CALL push(chosen_candidate_r_stack, member from list)
                 ELSE
                        discard element
                 END IF
                END IF
        END FOR
        IF phi<=-400
                phi = tan-1 ( next point_y / next point_x  )
        ELSE
                phi =tan-1 ( next point_y / next point_x  )+180
        END IF
        CALL Eliminate_from_list()
        RETURN phi
END PROCEDURE
```

The compromise, as viewed from the attacker's perspective, first intercepts a sample of the data that will be compromised at a future time. It is assumed that a partial capture will hold at least some portion of the underlying distribution. The underlying assumption is that the distribution is stationary and does not vary significantly in time. Using this assumption and the partial capture of information, it will be viable for the attacker to infer the underlying distribution, and therefore have an estimate of the clustering procedure outcome. This part of the process is shown in Figure 6, using the fifth and eight column from the first 10,000 samples of the Statlog (Shuttle) training data set. In this example, the attacker is assuming that the user of the K-means algorithm will use two centroids (shown as circles in Figure 6).

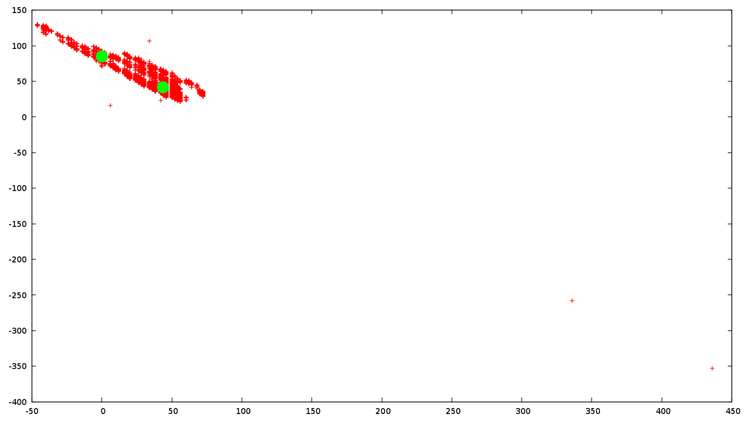


Figure 6. Initial cluster results with a 10K sample of the Statlog (Shuttle) data set

The next step of the process is to encode the payload, put it on the desired coordinate range, and insert it into the data set to see the behavior of the cluster. For this example, the chosen exploit is the Linux x86 ASLR deactivation payload. The result is shown in Figure 7, where it is shown that payload insertion is not enough to cause a substantial movement of the clusters. This would mean that if the attacker chooses to use as trigger the inter distance between the two clusters, he will need additional data points to trigger the exploit, or it might accidentally trigger if the measure presented is actually used.

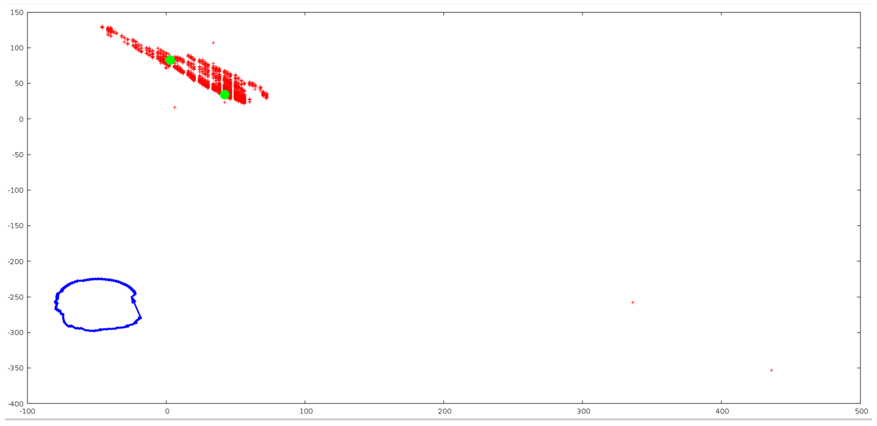


Figure 7. Cluster results with a 10K sample of the Statlog (Shuttle) data with payload injected

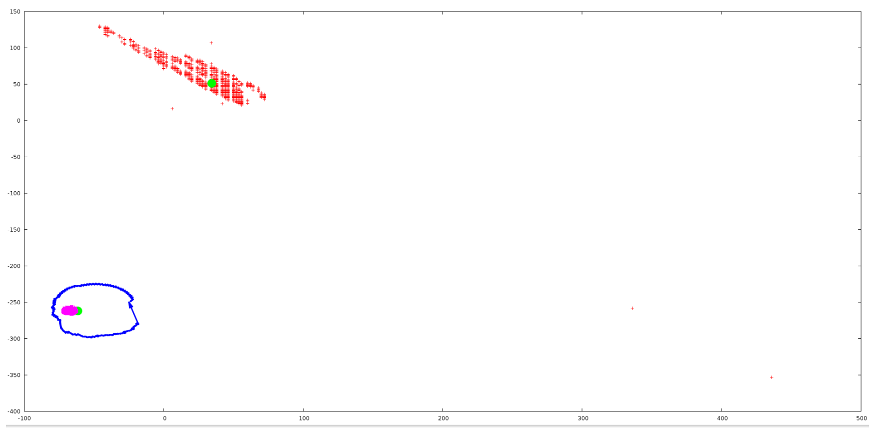


Figure 8. Cluster results with a 10K sample of the Statlog (Shuttle) data set with injected payload and 400 points

To solve the problem of the centroid distance, the attacker would need to insert random data points into the data sets, until he reaches a result such as the one shown in Figure 8. The increments were done in batches of 50 until reaching 400 data points. The random points were generated using the following center: -67.52 in the x coordinate and -261.31 in the y coordinate, with a maximum radius of 5. The radius size and the angles were randomly generated and then translated into x and y coordinates.

Careful analysis will have to be carried out on the part of the adversary under a scenario such as the one in the figures, where the inclusion of significant amount of data points and a substantial distance could raise suspicion. A careful analysis needs to be done by the attacker to minimize the distance needed to move the clusters, so that a reasonable amount of points generated can move the centroid to the desired coordinate.

As an additional consideration, when compromising other variants of K-means such as online K-means version, the attacker may leverage these additional data points to time the execution of the payload to his advantage. Under this scenario, the attacker must also estimate the learning rate range. This range can be reasonably be estimated by injecting specific data and viewing the response of the learning algorithm or its user. This scenario is plausible in intrusion detection, where the attacker can view a deny rule behavior activation on the part of the algorithm as part of the countermeasures of a filtering system.

Compromising Kohonen Neural Network

Kohonen Neural Network compromise presents several challenges that are not present in the K-means compromise. To understand the intricacies of the exploit, a review of the Kohonen Neural Network is given, followed by the compromise execution details.

The Self-Organizing Map (SOM) or Kohonen network (Kohonen, 2013, 1988) is a type of unsupervised Hebbian (Munakata & Pfaffly, 2004) network that originated from the early associative memory and adaptive learning networks, and can be used to solve clustering problems. A typical characteristic of the Kohonen derived from its relation to Hebbian networks is that it consists of just two layers: the input and the output or map layer.

The SOM neural network converts complex, generally high-dimensional data vectors with N features, into (usually) a low-dimensional map. The SOM consists of two fully connected layers of nodes: the input layer and the Kohonen layer, as shown in the figure below.

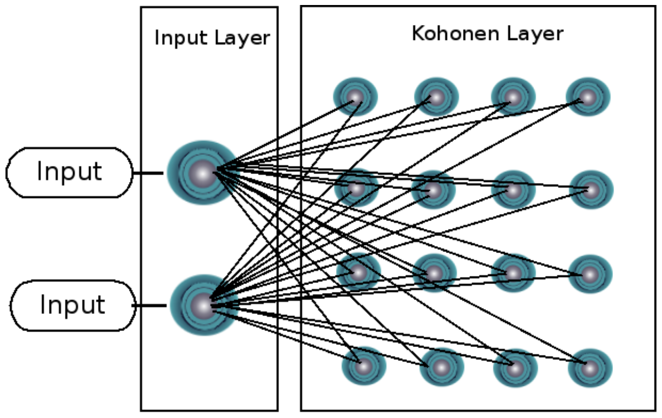


Figure 9. **Self Organizing Map**

The SOM can work as both a Maxnet and Minnet, but for this presentation, a Minnet will be assumed. The competitive learning of the SOM is related to the K-means algorithm, as explained in Bacao et al. (2005), where the update occurs only on the winning units of the Kohonen Neural Network.

The first phase consists on calculating the Euclidean distance given by $Distance_j$ for node j of the Kohonen layer:

$$Distance_j = \sum_{i=1}^{n} (Input_i - W_{old\,ij}) \tag{22}$$

Once the distance is calculated, then the next phase is to choose the winner through a Minnet process.

As part of the competitive scheme of the Minnet in a winner take all configuration, the network will suppress the contribution to the other nodes in the Kohonen layer output map, while "attracting" similar nodes. This suppression and ordering through similarity is manifested in the SOM through lateral feedback and the following learning rule:

$$W_{new\,ij} = W_{old\,ij} + c \cdot \eta\,(Input_i - W_{old\,ij}) \tag{23}$$

Where W are the weights, c is a learning coefficient and η is the neighborhood function. The original derivation of the formula is given in Kohonen (1982):

$$\frac{d\mu}{dt} = \alpha\,(\xi - \xi_b)\,\eta \tag{24}$$

where μ is the efficacy of the input weight, ξ is the presynaptic input, ξ_b is an effective background value, α is a proportionality constant, and η is the post synaptic triggering frequency. This equation yields:

$$\mu_{ij}(t+1) = \mu_{ij}(t) + \alpha\eta_i(t)\,(\xi - \xi_b) \tag{25}$$

with η as the neighborhood function that implements the lateral feedback, and in its simplest form, this function can assume binary values. It is set to 1 for those neurons being positively trained and zero otherwise. This equation *yields* the same result as Formula 23.

The next phase is to calculate the contribution of lateral feedback on the network. Several functions can be used to mode the lateral feedback of the SOM. The model can be given by a crude approximation of the rectangular function:

$$Rect(x) = \begin{cases} 0 & if\ \|t\| > \frac{1}{2} \\ \frac{1}{2} & if\ \|t\| = \frac{1}{2} \\ 1 & if\ \|t\| < \frac{1}{2} \end{cases} \tag{26}$$

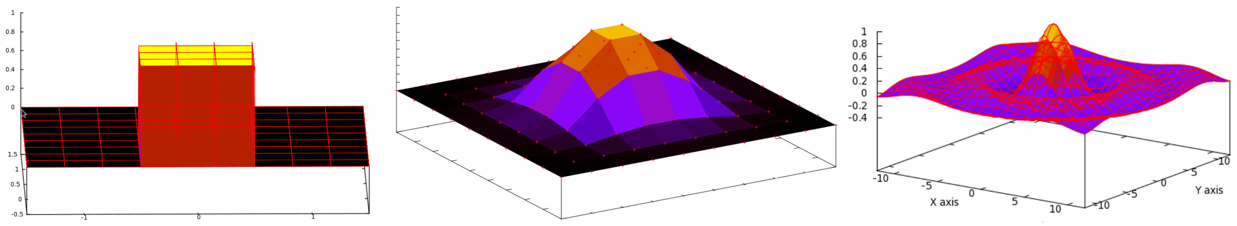
Another model implementation can be the Gaussian function:

$$f(x) = ae^{-\frac{(x-\mu)^2}{2c}} \tag{27}$$

or the Mexican Hat function (in one dimension):

$$f(x) = 1 - x^2 \cdot e^{\frac{x^2}{2}} \tag{28}$$

A graph of each of the functions is shown in the Figure 10 below.



a) Rectangular **b) Gaussian** **c) Mexican Hat**

Figure 10. **Lateral feedback function graphs**

The Kohonen Neural Network presents several challenges for the encoding and insertion of the payload of the malicious attack. While the attack can mimic the procedures of the K-means by substituting the term centroid for neuron, there is a lateral feedback to account for as an additional force that may disable the centroid distance as trigger, if not planned properly. Since what is usually of interest after the algorithm is used is not how well the data falls within the centroids, but how does the Kohonen layer and the weights behave, then this requires a change in payload encoding. In order to be stealthy, the attack must change, so that the payload is expressed in terms of these components. An alternative procedure is outlined in Algorithm 5.

Algorithm 5 Kohonen Trojan compromise procedure

```
1. Use a coordinate range on which one of the neurons may relocate after training, as a trigger
   for the Kohonen Trojan. The decoding uses the angle between the nodes of the Kohonen
   layer or the distances between the nodes to determine the payload $P'$.
   The decoding procedure depends on whether the distance or the angle is chosen.
2. Identify which nodes to be used as the malicious one, and quantify the error.
3. If necessary, calculate additional points to achieve the inter cluster distance or angle, and
   continue to step 4. Else, return to step 1.
4. Compromise the target by substituting the benign code with the compromised algorithm.
5. Inject the payload to achieve the predetermined objective.
```

Payload Encoder and Decoder for the Kohonen Neural Network Compromise

The Kohonen Neural Network attack differs from the encoding of the K-means payload. The main main difference in the algorithms is that the Kohonen algorithm acts as a dimensionality reduction process as mentioned above, by converting high-dimensional data vectors into (usually) a low-dimensional map. Another difference is that the clustering acts on multiple nodes of the Kohonen layer through lateral feedback, and the training may therefore influence more than one neuron which could be interpreted as more than one "centroid".

While the same encoding can be used to demonstrate the Kohonen compromise, several factors need to be considered. First, the encoding used in the K-means algorithm duplicates the amount of data points utilized for the compromise. This can be a drawback for the Kohonen algorithm, where the N*N Kohonen layer might be limited to a certain size and can compete with the benign data. To avoid this, two alternate encodings were employed for the compromise. A second factor to take into account is that the lateral feedback and random initialization of the clusters may destroy the order of the encoding, therefore rendering the compromise into a non functional one.

The first Kohonen payload encoder is shown in Algorithm 6 , where the compromise leverages the column values to insert the payload and the sequence order. The insertion of the payload is prety straight forward in that once the columns for the payload and the execution counter are chosen it is a matter of generating the payload and sequence value pair in lines 4 and 5 of the payload encoder algorithm for the SOFM. Since a new vector is being added and it cannot be presummed that the data will consist of only two columns, additional values are added through the function call in line 6. Finally, the function call of line 7 inserts the compromised vector into the dataset.

Algorithm 6 Payload encoder for SOFM compromise

```
1 PROCEDURE Encoder_SOFM(payload_values,column_compromise,
2                               column_counter, number_of_points_to_generate)
3         FOR i=1 to number_of_points_to_generate
4                 CALL insert_payload(payload_value,column_compromise)
5                 CALL insert_incremental_value(column_counter)
6                 CALL generate_rest_of columns_to_get_target_distance()
7                 CALL insert_additional_points()
8         END FOR
9 END PROCEDURE
```

The insertion of additional points in the encoder will help in shifting the weight training towards the specific coordinate point. The corresponding decoder (Algorithm 7) takes as input the final trained weights of the algorithm, the column where the sequence counter is located, the column where the payload is located and the sequence counter initial value. The last parameter is added since both the payload and the sequence number may be shifted to allow positioning within the weight space formed by the SOFM. Line 3 will initiate the reading of the weight matix, line 4 reads the values and line five will determine the resolution of the decoder (in decimal places) where a candidate is compared to the initial value. If the initial value falls within the tolerance of the decoder then it is recognized as the payload, its corresponding payload value is also extracted from the weights and the counter is incremented so that the routine can go to the next incremental candidate value.

Algorithm 7 Payload decoder for SOFM compromise

```
PROCEDURE Decoder_SOFM(SOFM_weight_array,column of_incremental_value,
                       column_of_payload, initial_counter_value)
  FOR i=1 to numberof rows -1
   (candidate 1, sequence value)=CALL extract_value (column of_incremental_value,column_of_payload)
   CALL determine_candidate(candidate 1, sequence value)
  END FOR
END PROCEDURE
```

The second attack aimed at the Kohonen Neural Network is more challenging than the first Kohonen compromise in its execution. The payload encoder inserts the compromised values as distance in the U-Matrix (Stefanovic & Kurasova, 2011) representation of the Kohonen Neural Network.

The attack is aimed not at the algorithm itself, but at the visualization of the algorithm's output. A graphical representation of what the algorithm works is shown in Figure 11.

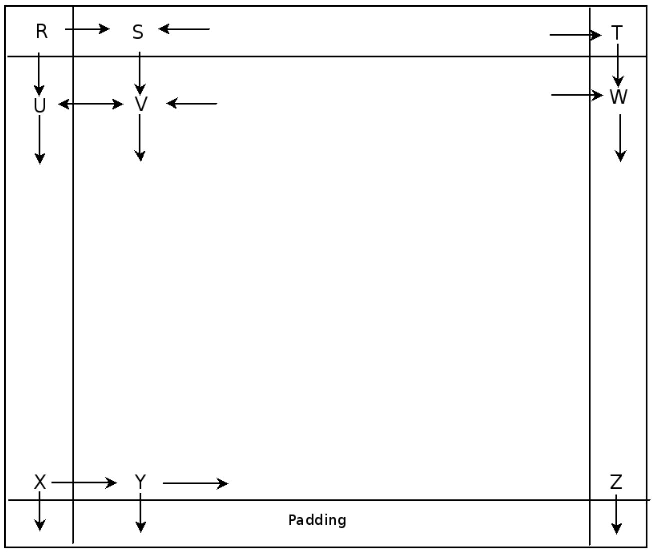


Figure 11. Graphical representation of the U-Matrix encoding compromise

Each of the entries in the U-Matrix will be constrained by the neighboring values. To construct the U-Matrix as is shown in Figure 11, the algorithm starts from the top left corner and goes through each row until finished. The entries can have free values which are indicated by the outgoing arrows. These values obey the formula:

$$y_{free} = (payload_{value} * number\,of\,sides) + (x_{current} * number\,of\,sides) - side\,values \quad (29)$$

The top left corner of the figure shows the initial value R which can be the start of the payload and is unconstrained by any value. The payload value of R can be obtained by finding the value, which divided by two, gives the desired payload result. The values in the first row, represented by *S,* have one free variable on the next row. The right hand value T is chosen randomly to complete the three way constraint of the S values.

The same process is repeated with the other values, taking into account the number of constraints. Once the first row is made, the center row follows, before taking the sides represented by U and *W*. Finally, the second to last row is over-constrained in that if padding is not supplied, an iterative process would have to take place to converge to a solution. Rather than using iterative converging solutions it was opted to use padding to simply compute the last values without the output constraint, allowing for a quick and efficient solution to the last row problem.

Algorithms 8 and 9 represents the pseudo-code for the encoder algorithm. Line 6 of the algorithm is responsible for generating the values for the first row values which are free parameters in the encoding scheme. These values can be prechosen or generated randomly, depending on the particular attack scenario. Lines 19 through 26 are designed to encode the first value which is the R value of Figure 11 using calls to functions given in Algorithm 10. The first step is to compute the *third value* using equation 29. This equation uses variables *first value* and *second value* which are the start of the exploit and are used to calculate the third value. Calls to get the two dimensional position into a one dimensional array and then store the values by assignning them to the U-Matrix *u_ encoding*

Algorithm 8 Payload encoder for SOFM U-Matrix compromise

```
PROCEDURE Encoder_SOFM(matrix_size_x,matrix_size_y,pad_value,first_value, second_value,begining,range)
 matrix_minus2_x=matrix_size_x-2
 matrix_minus2_y=matrix_size_y-2
 ;--generate random values for first row
 FOR x=0 to x<matrix_minus2_x
    values_first_xrow[x]=int(rand(range)) +begining
 END FOR

 coordinate =CALL load_attack_coordinate_array
 payload=CALL load_attack_payload_array
 CALL determine_if_padding_is_needed
 u_encoding[0]=first_value

 x_rand_ctr=0
 y_rand_ctr=0
FOR  y=0 to y< total entries in payload array
(offset_matrix_x,offset_matrix_y)= CALL calculate_offset(y,matrix_size_x,matrix_size_y)
  ;------corners----------
  IF((offset_matrix_y==0) AND (offset_matrix_x==0))
       third_value=(2*payload[0])-second_value+first_value+first_value
       u_encoding[0]=first_value
       array_coordinate=CALL get_array_position(matrix_size_x,1,0)
       u_encoding[array_coordinate]=third_value
       array_coordinate=CALL get_array_position(matrix_size_x,0,1)
       u_encoding[array_coordinate]=second_value
  END IF
 ;--------sides--------
  Else IF ((offset_matrix_y==0)AND (offset_matrix_x!=0)AND (offset_matrix_x!=matrix_size_x-1))
       x_right=values_first_xrow[x_rand_ctr]
       ;fill in the first row values of the U-Matrix one by one
       u_encoding[array_coordinate=CALL get_array_position(matrix_size_x, offset_matrix_y,
                          offset_matrix_x+1)]=x_right
       x_rand_ctr++
       x_left=u_encoding[array_coordinate=CALL get_array_position(matrix_size_x, offset_matrix_y,
                          offset_matrix_x-1]
       x_curr=u_encoding[y]
       y_down=(payload[y]*3)+(x_curr*3)-x_left-x_right
       u_encoding[array_coordinate=CALL get_array_position(matrix_size_x, offset_matrix_y+1,
                          offset_matrix_x]=y_down

       IF offset_matrix_x==matrix_size_x-2
          x_curr=u_encoding[array_coordinate=CALL get_array_position(matrix_size_x,
              offset_matrix_y, matrix_size_x-1)]
          payload_value=payload[array_coordinate=CALL get_array_position(matrix_size_x,
              offset_matrix_y, matrix_size_x-1)]]
          x_left =u_encoding[array_coordinate=CALL get_array_position(matrix_size_x,
              offset_matrix_y, matrix_size_x-2]
          y_down=(payload_value*2)+(x_curr*2)-x_left
          u_encoding[array_coordinate=CALL get_array_position(matrix_size_x,
              offset_matrix_y+1, matrix_size_x-1]=y_down
        END IF

  END IF
```

Algorithm 9 Continuation of payload encoder for SOFM U-Matrix compromise

```
Else IF((offset_matrix_y!=0) AND (offset_matrix_y!=matrix_size_y) AND (offset_matrix_x!=0)
    AND (offset_matrix_x!=matrix_size_x-1))
   x_curr=u_encoding[y]
   y_up=u_encoding[ array_coordinate=CALL get_array_position(matrix_size_x, offset_matrix_y-1,
           offset_matrix_x]
   x_left=u_encoding[array_coordinate=CALL get_array_position(matrix_size_x, offset_matrix_y,
           offset_matrix_x-1]
   x_right=u_encoding[array_coordinate=CALL get_array_position(matrix_size_x, offset_matrix_y,
           offset_matrix_x+1]
   y_down=(payload[y]*4)+(x_curr*4)-x_left-x_right-y_up
   u_encoding[array_coordinate=CALL get_array_position(matrix_size_x,offset_matrix_y+1,
           offset_matrix_x]=y_down

     IF (offset_matrix_x==matrix_size_x-2){
      ;------- first entry of the row
        x_curr=u_encoding[array_coordinate=CALL get_array_position(matrix_size_x,
            offset_matrix_y, 0]
        payload_value=payload[array_coordinate=CALL get_array_position(matrix_size_x,
            offset_matrix_y, 0]
        y_up=u_encoding[array_coordinate=CALL get_array_position(matrix_size_x,
            offset_matrix_y-1, 0]
        x_right =u_encoding[array_coordinate=CALL get_array_position(matrix_size_x,
            offset_matrix_y, 1]
        y_down=(payload_value*3)+(x_curr*3)-x_right-y_up
        u_encoding[array_coordinate=CALL get_array_position(matrix_size_x,
            offset_matrix_y+1,0]=y_down

      ;------- last entry of the row
        x_curr=u_encoding[ array_coordinate=CALL get_array_position(matrix_size_x,
             offset_matrix_y, matrix_size_x-1]
        payload_value=payload[ array_coordinate=CALL get_array_position(matrix_size_x,
             offset_matrix_y, matrix_size_x-1]
        y_up=u_encoding[array_coordinate=CALL get_array_position(matrix_size_x,
             offset_matrix_y-1,  matrix_size_x-1]
        x_left =u_encoding[array_coordinate=CALL get_array_position(matrix_size_x,
             offset_matrix_y, matrix_size_x-2]
        y_down=(payload_value*3)+(x_curr*3)-x_left-y_up
        u_encoding[array_coordinate=CALL get_array_position(matrix_size_x,
             offset_matrix_y+1,matrix_size_x-1]=y_down
    END IF

  END IF

 END FOR

END PROCEDURE
```

The process of filling the entire matrix follows the same same format as the calculation for the R value of value of Figure 11 with the exception that as the matrix progresses the algorithm has to extract previous calculated values. each one of the quadrants in the Figure 11 is covered by exceptions specified in IF conditionals. The routine keeps executing through the for loop until all values are entered into the U-Matrix.

Algorithm 10 Support procedures for payload encoder for SOFM U-Matrix compromise

```
PROCEDURE  calculate_offset (calc_value, x_value)
 offset_value_x= calc_value% x_value;
 offset_value_y= calc_value/ x_value;
 return( offset_value_x,int( offset_value_y));
END PROCEDURE


PROCEDURE  get_array_position (y_pos, x_size, x_pos)
  array_pos= y_pos* x_size+ x_pos;
  return  array_pos;
END PROCEDURE

PROCEDURE  rand_range
 return int(srand( y -  x)) +  x;
END PROCEDURE

PROCEDURE determine_if_padding_is_needed
 mat_size= matrix_size_x* matrix_size_y;
 IF tot< mat_size
   last_value_ctr= coordinate[ tot-1]+1;
   diff= mat_size- tot;
  FOR  x=0 to x< diff
    push(payload, pad_value);
    push(coordinate, last_value_ctr);
    last_value_ctr= last_value_ctr+1;
  End FOR
 Else IF $tot>$mat_size
  die "matrix size is too small for payload"
 END IF
END PROCEDURE
```

The decoder shown in Algorithm 11 is just the formula for the U-Matrix which for the center values is given by the formula:

$$Payload_{value} = \frac{(x_{right} - current + x_{left} - current + X_{up} - current + X_{down} - current)}{4} \tag{30}$$

An additional element to pass to the decoder is the location of the sub rectangle that composes the payload. This will give the position of the payload within the Kohonen layer and will instruct the decoder on where to look for the sub matrix. As with the Encoder the algorithm is divided into the sections of Figure 11 by the IF conditions within the algorithm. The *For* loops in lines 2 and 3 iterate through each element of the array and depending on the position within the matrix it will process Equation 30 with the appropriate number of sides and store it in a variable distance which can then be used as the value of the payload.

Algorithm 11 SOFM payload decoder U-Matrix compromise

```
PROCEDURE Decoder_SOFM(matrix_size_x,matrix_size_y, coordinate, payload)
 For y=0 to y<matrix_size_y
  For x=0 to x<matrix_size_x
   array_coordinate=get_array_position matrix_size_x,y,x
   current=payload[array_coordinate]
   IF  x==0 AND y==0
        x_right=payload[ array_coordinate = CALL get_array_position(matrix_size_x,y,x+1)]
        y_down=payload[ array_coordinate = CALL get_array_position(matrix_size_x,y+1,x)]
        dist=(x_right-current + y_down-current)/2
   End IF
   IF  x!=0 AND x!=matrix_size_x-1 AND y==0
        x_right=payload[array_coordinate = CALL get_array_position(matrix_size_x,y,x+1)]
        x_left=payload[array_coordinate = CALL get_array_position(matrix_size_x,y,x-1)]
        y_down=payload[ array_coordinate = CALL get_array_position(matrix_size_x,y+1,x)]
        dist=(x_right-current+x_left-current+y_down-current)/3
   End IF
   IF  x == matrix_size_x-1 AND y==0
        x_left=payload[array_coordinate=CALL get_array_position(matrix_size_x,y,x-1)]
        y_down=payload[array_coordinate=CALL get_array_position(matrix_size_x,y+1,x)]
        dist=(x_left-current + y_down-current)/2
   End IF
   IF  x == 0 AND y!=0 AND y!=matrix_size_y-1
        x_right=payload[array_coordinate = CALL get_array_position(matrix_size_x,y,x+1)]
        y_up=payload[array_coordinate = CALL get_array_position(matrix_size_x,y-1,x)]
        y_down=payload[array_coordinate = CALL get_array_position(matrix_size_x,y+1,x)]
        dist=(x_right-current + y_up-current + y_down-current)/3;
   End IF
   IF  x!=0 && y!=0 AND x!=matrix_size_x-1 AND y!=matrix_size_y-1
        x_right=payload[ array_coordinate=CALL get_array_position(matrix_size_x,y,x+1)]
        x_left=payload[array_coordinate=CALL get_array_position(matrix_size_x,y,x-1)]
        y_up=payload[array_coordinate=CALL get_array_position(matrix_size_x,y-1,x)]
        y_down=payload[array_coordinate=CALL get_array_position(matrix_size_x,y+1,x)]
        dist=(x_right-current + x_left-current + y_up-current + y_down-current)/4
   End IF
   IF  y!=0 AND x==matrix_size_x-1 AND y!=matrix_size_y-1
        x_left=payload[array_coordinate=CALL get_array_position(matrix_size_x,y,x-1)]
        y_up=payload[array_coordinate=CALL get_array_position(matrix_size_x,y-1,x)]
        y_down=payload[array_coordinate=CALL get_array_position(matrix_size_x,y+1,x)]
        dist=(x_left-current + y_up-current + y_down-current)/3
   End IF
   IF  x==0 AND y==matrix_size_y-1
        x_right=payload[array_coordinate=CALL get_array_position(matrix_size_x,y,x+1)]
        y_up=payload[array_coordinate=CALL get_array_position(matrix_size_x,y-1,x)]
        dist=(x_right-current + y_up-current)/2;
   End IF
   IF  y==matrix_size_y-1 AND x!=0 AND x!=matrix_size_x-1
        x_right=payload[array_coordinate=CALL get_array_position(matrix_size_x,y,x+1)]
        x_left=payload[array_coordinate=CALL get_array_position(matrix_size_x,y,x-1)]
        y_up=payload[array_coordinate=CALL get_array_position(matrix_size_x,y-1,x)]
        dist=(x_right-current + x_left-current + y_up-current)/3
   End IF
   IF  y==matrix_size_y-1 AND x==matrix_size_x-1
        x_left=payload[array_coordinate=CALL get_array_position(matrix_size_x,y,x-1)]
        y_up=payload[array_coordinate=CALL get_array_position(matrix_size_x,y-1,x)]
        dist= (x_left-current + y_up-current)/2
   End IF
  End For
End For

END PROCEDURE
```

Compromising Adaptive Resonance Algorithm

Adaptive Resonance Theory, developed by Stephen Grossberg, is a family of supervised and unsupervised algorithms (Carpenter & Grossberg, 1998):

- ART1- is an unsupervised model that is "capable of self-organizing, self-stabilizing, and self-scaling its recognition codes in response to arbitrary temporal sequences and arbitrarily many input patterns of variable complexity" (Carpenter & Grossberg, 1987b).
- ART2- is the "class of adaptive resonance architectures which rapidly self-organize categories in response to arbitrary sequences of either analog or binary input patterns" (Carpenter & Grossberg, 1987a).

The current work will concentrate only on the ART1 algorithm that consists of two layers that are fully connected and use binary representation for both layers. The layers are called F1 and F2, for the first and second layers, respectively. There are forward connections from each neuron of *F1* to each of the neurons at the *F2* layer via forward weights WF_{ij}, as shown in Figure 12. The same is done from all neurons at *F2* to neurons at *F1* via WB_{ij}. The initial architecture starts as in Figure 12 a), where there are no neurons at the *F2* layer. In addition, there are two gain subsystems that control the neural network behavior. The network is initialized to *F2 = 0*, if no previous data has been shown to the network. The initial values of WF_{ij} are chosen randomly, such that:

$$0 < WF < \frac{L}{(L-1)+|M|} \tag{31}$$

where M is the number of nodes representing the input vector, and the constant $L > 1$. The vigilance parameter ρ is set to the desired value. The weights of *WB* are all set to 1. A high vigilance value will yield a highly selective clustering, producing clusters of tightly coupled members. A lower vigilance parameter value will results in more general clusters, and will be less discriminative.

The steps to execute the neural network are as follows:

Step 1: For each input, do:

$$f1 \rightarrow f2 = \sum_{i=1}^{inputs} WF \cdot f_{i(0,1)}(X,\, gain,\, feedback) \tag{32}$$

The input function $f_{0,1}$ takes as arguments three parameters: the inputs *X,* the gain, and the feedback. The gain is the result of two individual gain mechanisms: Gain 1 and Gain 2. Gain 2 is the logical *or* of the inputs *X,* such that:

$$Gain2 = \begin{cases} 1 & if\, or\, X \neq 0 \\ 0 & if\, or\, X = 0 \end{cases} \tag{33}$$

The output of Gain 1 is given by:

$$Gain1 = \begin{cases} 1 & if\, inputs \neq 0\, and\, T = 0 \\ 0 & if\, x < \sigma \end{cases} \tag{34}$$

where T is the logical *or* of the outputs *t*. The feedback is the amount of the $node_j$ at *F2* with the corresponding weight that points to the input node being calculated. The feedback is more formally expressed as:

$$feedback_i = \sum_{j=1}^{M} WB \cdot t \tag{35}$$

From the three inputs: X, $gain$, $feedback$, the activated nodes will be the ones that pass the 2/3 rule, in which at least two of the three components must be present in order to activate. This means that at least the gain or the feedback must activate the neuron.

Step 2: From this, select the output node that has the biggest output, such that:

$$Max\,(f1 \rightarrow f2) \tag{36}$$

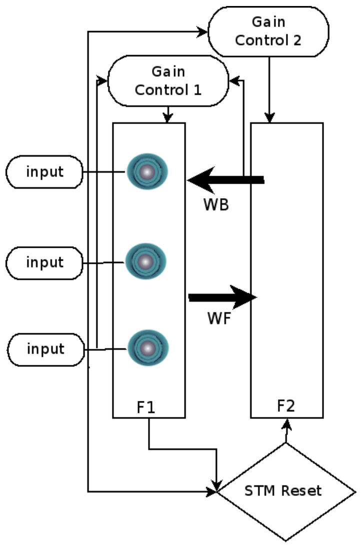
Step 3: The selected node will then go through the vigilance test, which evaluates the closeness of the input pattern to the stored "ideal pattern of the node":

$$\frac{\sum_{i=1}^{inputs} WF \cdot f_{0,1}(X,\, gain,\, feedback)}{\sum x} > \rho \tag{37}$$

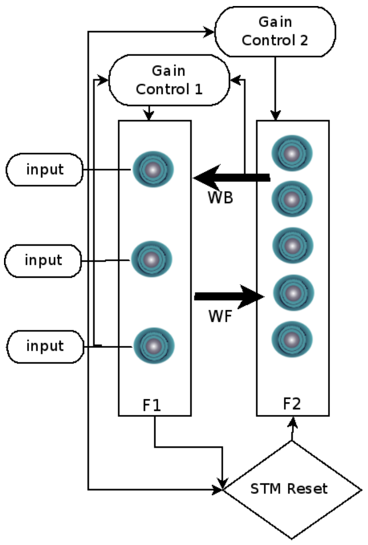
Step 4: If the node fails the vigilance test, then mask the node as unavailable, and pick the runner up. Repeat this step, until a node passes the vigilance test. Should all neurons fail the test, then create a new node.

Step 5: For the winner node, do the training with:

$$WF_{ij} = \frac{L \cdot f_{i(0,1)}(X,\, gain,\, feedback)}{L - 1 + \sum f_{(0,1)}(X,\, gain,\, feedback)}\; and\, t_{ij} = f_{i(0,1)}(X,\, gain,\, feedback) \tag{38}$$



a) ART1 starting state



b) ART1 with nodes in *F2*

Figure 12. **Adaptive Resonance Theory network**

Payload Encoder and Decoder for the Adaptive Resonance Algorithm Compromise

The attack on Adaptive Resonance differs substantially from the previous two attacks in that the processing carried out by ART1 is binary in nature. The encoding of the payload is broken into two main segments: the payload itself and an ordering part. The ordering part is essential in case the strings arrive at the algorithm in different order. Figure 13shows three possible alternatives to encode the payload and the order.

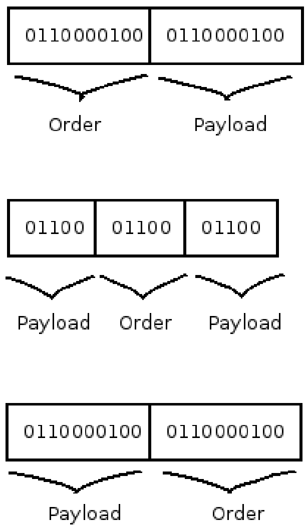


Figure 13. Encoding layout of payload and order

The encoder for the Adaptive Resonance payload is given in Algorithm 12.

Algorithm 12 Payload encoder for Adaptive Resonance compromise

```
PROCEDURE Encoder_ART(list_of_hex_characters_payload,sequence,data_sample)
           CALL insert_payload(payload_value,column_to_compromise)
           CALL insert_incremental_value(column_counter,column_to_compromise)
           CALL Insert_zero_value_padding(column_to_compromise)
           CALL convert_dataset_to_binary(compromised_data_sample)
END PROCEDURE
```

One particular difference in the encoding process using Algorithm 12 is that it requires more trial and error on the part of the attacker to achieve a particular payload encoding. The formula for the division of clusters may yield multiple compromised clusters which may make the attack cumbersome in nature. However, the decoder for the ART1 payload is simpler in nature, than the other decoders mentioned thus far, and is given in the Algorithm 13.

Algorithm 13 Payload decoder for the Adaptive Resonance compromise

```
PROCEDURE Decoder_ART(sequence column_id, dataset column,starting_value ,payload_size)
        CALL Sort_column(dataset column)
        CALL Identify_start_value(starting_value, sequence _column)
        ;returns offset to starting_value in sequence column
        FOR j= starting_value_offset to payload_size value
                IF  sorted_column_value_at_offset == i
                        CALL extract_compromised Column_value()
                END IF
        END FOR
END PROCEDURE
```

The selected encoding for the tests carried out for this thesis was the order-payload sequence encoding. While in other compromises random numbers can be generated based on information from the sample distribution, in Adaptive Resonance this can be counter productive. The additional data elements inserted into the columns can lead to a detrimental discrimination in Adaptive Resonance. For the tests carried out a zero value entry in each binary value was selected as method of padding the payload.

Experimental Setup for the Attack Simulation

To carry out the scenario outlined above, it assumes that the attacker has a well established methodology for analyzing the behavior of the data under study. The attacker must also have a measurement of effectiveness that maximizes the probability of success of the attack, while maintaining the clusters as close as possible, to avoid arousing suspicion.

An additional requirement is that the attacker must find a way to simulate more data for the attack scenario, to test the payload injection and triggering of the attack. If he does not have access to a sample, then the attacker could assume a random distribution. However, this simulation works under the reasonable assumption that an attacker has access to intercepted or obtained data sample.

Another reasonable assumption is that the intercepted or obtained data is just a sub-sample of the underlying distribution that will be processed by the algorithm. A possibility for the attacker is to leverage the obtained sample to generate additional samples. While bootstrapping (P. Cohen, 1995) can be an effective way to generate additional training sets, it does not necessarily meet the particular requirements for the evaluation of the attack's effectiveness. An alternative is to generate the data set as bootstrap, but with an additional generation consisting of perturbed data points. From this, two alternatives can immediately be utilized to test an attack's effectiveness:

- No addition of points to serve as a baseline.
- Based on the datapoints, use a normal distribution to distribute the random points.

The second option is based on obtaining the mean of each of the dimensions of the data points, and then use the first, second, and third standard deviations to create the additional random points.

The proposed experimental setup is outlined to simulate the attacker's setup and actual attack:

1. Divide into training and testing using a 20-80 rule or similar ratio, respectively. This will simulate reasonable real scenarios of an attacker capturing partial information gathered by the user of the unsupervised learning algorithm. If the data set contains a testing and training set, they must meet these parameters. If training set exceeds this ratio, elements will be eliminated randomly from the training set until the ratio is met.
2. Calculate the minimum, maximum, mean and standard deviation from the obtained set to be used as training.

3. From the selected set used as training, generate a training set using the bootstrap method and generate a second one to use as the baseline. For the Bootstrap data set, randomly generate additional n elements on the first, second, and third standard deviations using the mean deviation for the first technique, using 68–27–5.7 percentages of the standard deviations percentages of this distribution. A baseline of no additional elements will first be run, followed by the two techniques generating the random elements, using the total number of elements of the test set. This particular number of samples is given under the assumption that the attacker can estimate the number of samples to be processed, but does not have access to the particular set of elements to be processed by the algorithm at testing.

4. Insert the malicious data into the benign data set serving as training.
5. Process the payload through the decoder, to see if it decodes.
6. Perform the attacker's analysis of the ULT Density Problem. Evaluate the result taking into account additional constraints such as stealthiness of the candidate solution and the ULT density measurements.
7. Tweak the solution until satisfied or until a candidate solution emerges from the training runs.
8. Repeat the procedure using different data sets and different parameters.
9. Run the malicious payload against the test set, record the results as success or failure on triggering the payload, and whether the decoder succeeds or not.
10. Carry out significance tests for the results obtained.

The step 10 of the proposed experimental method makes two measurements that can benefit from the quantification and use of significance test (Wackerly et al., 2002). The first one will evaluate the use of a particular attack on a particular shared data set between algorithms. The encodings that are used on each of the algorithms vary substantially and there is a need to know if a particular encoding is more efficient than another. The assumption tested uses three particular algorithms, four different malicious payload encodings, and different data sets for the tests. Under the described test conditions where three classifiers are used and no assumptions can be made to justify ANOVA (Demsar, 2006), statistical significance testing is done using the Friedman nonparametric test.

The second measurement carried out is the use of a baseline of just the obtained data, set as opposed to generating additional samples to approximate the actual scenario of the compromise. The assumption used to justify the test of significance is that while there are three different algorithms, the comparison will be made under the same parameters and the same data sets. The only difference between the tests will be the addition of samples for the non baseline test. The parametric test under this assumption would be equivalent to two different classifiers (Demsar, 2006). Under these conditions, the statistical evaluation selected is the Wilcoxon signed ranked test.

To standardize results for the data sets for all tests with the exception of the Adaptive Resonance algorithm, the distances of the payloads were arranged so that either:

1. The complete payload stays outside the 1st standard deviation in at least one of the tests.

2. Half of the payload values fall within the 3rd and standard deviation in at least one of the tests.

3. The payload falls completely within the right tail of standard deviation and the minimum value on the left tail of the distribution.

This standardization will help in making a consistent result tabulation for the hypothesis testing. The Adaptive resonance algorithm exception will be discussed below.

For the K-means, the fourth step will be based on one of the three case scenarios above, and consists on generating a random coordinate, which will become the malicious cluster centroid's initial placement. The attacker will calculate the DS, the DCD, and Dy_{Trojan} for the setup on step 6. Optimization will be tested to see if the trigger of the payload on step 5 is adequate. This will be adjusted repeatedly, as stated in step 7 and as needed, until the attacker is confident that he has obtained a viable solution. The process will be repeated for three random coordinates.

The process will be repeated using the KEGG Metabolic, Statlog (Shuttle), and Water Treatment data sets, using payloads 2, 3 and 7 from Table 2. The KEGG Metabolic data set the data was divided into a 20-80 ratio which is 10,683 for training and 42,730 of the 53,413 samples in that data set. The Statlog (Shuttle) data set was divided into 2,900 random samples chosen from the training file and the full test data of 14500 samples for the test run. Finally Water Treatment data set was processed using only Fpv Close,High, Bypass, Bpv Close, and Bpv Open columns with 105 samples for training and 422 samples for testing.

All the 18 tests carried out for the K-means algorithm were done using 2 clusters on the first two columns of the data sets. In the final step, the tests were run and significance tests were made to see if there was a difference between the ratios.

The Kohonen Neural Network attack testing will be similar to the K-means compromise attack with some changes. On step 1, the percentage will be approximately 27-73 ratio due to the small sizes of the selected data set. On step 4, the attacker will use the generated malicious data to move the nodes of the Kohonen layer into the appropriate coordinates. From there, the attacker will calculate the DS, the DCD, and DY for the setup. The measurements will be accompanied by the payload tests in step 4 to see if the decoder functions properly. The process of the Kohonen compromise is repeated and adjusted to find the number of elements that make the nodes move to the precise coordinates. The DS will determine if an actual error occurs; and the DCD and DY will help in determining if the chosen distance has a probability of failure. These values will help determine the amount of tolerance for the decoder.

The process of improving the Kohonen layer movement of points is refined by repeating the process by using different random coordinates and 2 of the payloads to be minimized. The malicious payload is then used against the test set, the results recorded as success or failure on triggering the payload, and whether the decoder succeeds or not. The results of these tests will then be validated using tests of significance using the non parametric Friedman test on preselected ratios, to see if the increase in ratios is significant. The Null hypothesis states that there is no difference between using different ratios.

The selected data sets for the Kohonen Neural Network attack simulation are: KEGG Metabolic, Power Consumption, and Water Treatment using the payloads 5 and 6 from Table 2. The data sample sizes for the Kohonen algorithm were restricted to random samples sizes of 200 for the training set, but were limited to 560 samples for the testing in the KEGG Metabolic and Power Consumption data sets due to processing time restrictions. For the water treatment data set, the 105 sample size for the training set was used alongside 422 instances for the test set.

The Adaptive Resonance experimental setup somewhat deviates from the other two algorithms due to its binary nature. Given the particular clustering of Adaptive Resonance, a first run will test the point were zero error is made at .1 vigilance parameter setting; and then a Gaussian number generator will be used to select 2 numbers around this value to see how the vigilance varies. If the samples vary, then the one with the highest DCD will be chosen as representative for the test run.

The offset of the payload from the benign data is determined on zero based error given the vigilance parameter. The DS reads 0 because there are no errors in the payload in a discrete cluster that has been fully separated. The DY will also reach a steady state within the same sample and depends on the volume of the cluster being described. This process is done using two of the specified payloads to be minimized. Besides the difference due to the measurements, the rest of the procedure steps are the same as the ones described above for the use of the KEGG Metabolic and Statlog (Shuttle) data sets, using payloads 8 and 4 from Table 2. The data set sizes and column selections for the Adaptive Resonance training and testing followed the same specifications as the K-means algorithm.

Quantifying and Scoring the Results of the Training and Testing Data

This section is aimed at explaining the quantifying and scoring of the results reported in the experimental results chapter and Appendices B through E. The section is divided into three parts. The first part is an overview of the evaluation format of the appendices, the second is the training evaluation, and finally, the evaluation of the test data.

The evaluation format of the experimental runs given in the appendices is provided in a table format, and is divided into the training results and the test results. The table section belonging to the training starts with the sample's basic statistics such as min/max values, mean and standard deviation. The sample statistics display basic measures from which to compare the samples of the training and the test runs. The statistics reported for the sample consist mainly on the columns that are used in the compromise. The other tables are not reported, since depending on the attack, they can be considered noise or simply do not provide any measurable difference to the results.

The sample statistics are followed by the three samples' positions with regards to the sample space where the compromise is taking place. The positions are usually given in terms of the offsets to start the payload and sequence order. The sequence is located in the first column of the selected data set. The second coordinate is that of the value used as offset to start the payload encoding embedded in the second column of the selected data set. Afterwards, each sample's use of additional points, additional payload data elements, DS, DCD, DY_{Trojan}, and DY_{Benign} are reported. The number of additional points column is not utilized if the runs are part of the baseline training. The additional payload column is not utilized in Adaptive Resonance, instead it is utilized to report the specific vigilance parameter used for the training run.

The DS, DCD, DY_{Trojan}, and DY_{Benign} are reported for each training run and utilized in the evaluation of the training run. The only measurement that is excluded in the evaluation criteria for all runs is the DY_{Benign}. The rationale for the exclusion lies in that while this property is useful in providing a sense of the training set distribution, it was found not to be an effective indicator as the other measurements, due to the low ratio of intercepted messages utilized in the process. As the adversary gathers more information about the final distribution, then the measurement can become more reliable and more important in the decision process.

The testing report is similar to that of the training starting with the testing data's basic statistics. The statistics are followed by the DS, DCD, DY_{Trojan} and DY_{Benign} measurements. Finally, a confusion matrix for the test run is provided to quantify the error in the test run. While the DS provides a basic error reporting, the confusion matrix provides a more detailed information on where the error occurred.

The measurement's interpretation taken during the training phase of the compromise will depend on the particular goal of the attacker. If the attacker's goal is to provide stealthiness, then particular care needs to be taken in interpreting the ULT DS, DCD, and DY measurements. These formulas for documenting the ULT measurements are aimed at discriminating the payload from the benign data. Under such assumption, the cluster measurements will tend to favor compromises in which the attack may be considered an outlier and therefore draw attention. To take into account stealthiness, a tradeoff has to be made between guaranteeing the execution of the payload and the stealthiness factor of the attack. The additional payload represents additional data elements that are injected to bias the centroids towards the desired outcome.

For the K-means and Kohonen algorithms training the following scoring of each individual run was done using the information shown in Table 3:

	DS,	DCD,	DY	Additional Payload	Distance to Mean
Best Result	Highest	Lowest	Highest	Lowest	Closest
Highest Points Awarded	3	3	3	3*1.5	3*2
Lowest Points Awarded	1	1	1	1*1.5	1*2

Table 3: K-means and Kohonen scoring criteria

The scoring criteria tries to balance the measurement's objective of cluster discrimination and the need to remain less detectable. The two additional evaluation criteria of awarding points for compromising the data set with the least amount of additional malicious points and the closeness to the mean bring the desired tradeoff between cluster discrimination and detectability. The evaluation criteria also took into consideration when two runs had the same outcome on each individual measurement. Under such circumstance, a value of 1.5 or 2.5 was awarded, depending on whether the tie was for first or second place.

An additional consideration in the K-means algorithm run evaluation was the misclassification assignment. The evaluation consisted on granting a misclassified status of the benign cluster if both centroids fell inside the payload circle. Similarly, if both centroids fell outside the circle of the payload, then the payload is misclassified as benign. The rationale for this evaluation criteria consisted in the algorithm's goal to move the centroids and not the data, therefore the evaluation rested not on the data location, but on the centroids' location. While this evaluation criteria was the selected one, there are other criteria that can be used, such as quantifying the data points that were misplaced based on the Euclidean distance to the centroids. The complication would then be on determining which centroid is considered benign or malicious in a predetermined fashion.

For the Kohonen algorithm, the rules of classification were made based on a threshold value. If the benign neuron or weight fell within the integer value of the encoding for the payload, then the point was misclassified as malicious. In contrast, if a point did not reach convergence, then it was misclassified as benign.

As opposed to the K-means and Kohonen algorithm, the Adaptive Resonance candidate selection algorithm varied considerably. As previously mentioned, since Adaptive Resonance is based on a binary classification without preselection of the number of clusters, the measurements selected for the other two algorithms would not give accurate results. As an alternative, the procedure for Adaptive Resonance was done by choosing the closest distance to the cluster, since the encoding will be affected and turned into a longer binary sequence that can be easily detectable upon inspection. The selection criteria was based on randomly selecting the vigilance parameter with a value between 0.01 and 0.10 , a value at 0.10, and a value between 0.10 and 0.20. The winner was selected based on which value obtained the zero result with the lowest offset in the payload column. This minimum offset provided the encoding values tested on a randomly selected vigilance parameter.

Discussion of Possible Real World Scenarios

This provides a discussion of further considerations of the attack under a real world scenario. The discussion focuses on initial compromise scenarios, distribution channels, and advantages of certain programming languages in hiding the compromise. Each of the different unsupervised learning algorithms is used under certain scenarios, and the implementations of the ULT will vary to accommodate changes in environment.

The ULT attack instances are considered to be possible under two particular scenarios which the simulations carried out try to capture. The first scenario considers an attack by an internal agent that has access to the programming or hardware configuration of an organization that uses learning algorithms for the analysis of its data. This scenario reinforces the notion of Kerckhoffs' Principle discussed in the review of literature. Under this scenario, the attacker has complete access to the information, providing a scenario in which the end user of the algorithm cannot not rely on expectations of secrecy, since the information is shared or can even be generated by an attacker.

The second scenario contemplates an attacker that distributes the software himself.

The attacker will either portray himself as a distributor of the software or modify an existing distribution channel for an existing unsupervised learning software. Again, under this scenario, it is the attacker who is providing the software and can alter the software and even the supplied data, if the conditions are appropriate.

The capacity of the end user noticing the compromises will depend on the auditing done periodically on the source code and the experience of the user with the data that he processes. In terms of the programming, the detection will depend on the specific activation routine and the lines of code necessary to carry out the decoding and execution of the payload. Some implementations, such as with a compiled language programming, can be extremely difficult to detect once the application is compiled. Interpreted languages and accessibility to source code will be more amenable to evaluation.Also, a forensic analysis may yield discovery of the compromise.

Depending on the end user's experience and the particular activation and decoding used in the ULT, if the attacker also needs to unbalance the data set excessively through additional compromised data points to carry out the attack, the user may be able to notice the discrepancy with previous results. As the benign data and the payload deviate from the usual data patterns for the particular problem domain, the easier it will be to spot the attack.

Resources

The resources used for the dissertation work consist of computing and software resources, and data sets processed. Each of the three resource types are discussed below.

The computing resources used to carry out the proposed work consist of a workstation with a Pentium i7 processor, 9 GB RAM, and a 500 GB storage capacity.

Necessary software resources are: preprocessing and insertion of payload, unsupervised learning programs, and post processing software to quantify and visualize the results. The unsupervised learning software was programmed in C language and compiled with the GCC compiler, running on Fedora Linux. The preprocessing and insertion of the payload into the unsupervised learning algorithms was programmed using PERL for easy of modification and evaluation. Visualization primarily consists of custom scripts with GNUPlot and LibreOffice Calc charts.

Summary

This chapter has introduced the unsupervised Learning Trojan Density Problem, where the attacker does not have control of a training set. This implies that the parameters of the attack have to be estimated, instead of learned, and the problem is now expressed in terms of the density estimation of the distributions. Under this scenario, the chapter has introduced the question of whether an unsupervised algorithm is capable of discriminating between unlabeled malicious samples and the benign data.

The material covered several techniques that the attacker can use to measure the progress and the quality of the attack. As the narrative continues, a cluster identification sub problem is presented, which inquires the reliable identification of the malicious cluster from those identified by the algorithm as benign.

The chapter also covered the payload activation and encoding strategy used in these types of attacks. The chapter narrative then provided the three algorithms and the four data sets to be used in the experiments. Eight payloads were specified to carry out the attacks. The algorithms, data sets, and payloads in the provided experimental setup produce the following: 18 test runs on the K-means algorithm, 12 on the Kohonen Neural Network, and 12 runs on ART-1. The chapter included the procedures and the particular encodings utilized for each of the algorithms, and provided the necessary experimental setup and resources to carry out the experiments under this dissertation.

Chapter 4

Results

Experimental Results for K-Means

This section provides a discussion of the results obtained from the experiments carried out with the K-means algorithm on the KEGG Metabolic, Statlog (Shuttle), and Water Treatment data sets, using the Linux_x86 Polymorphic ShellCode - setuid(0)+setgid(0)+add user 'iph' without password to _etc_passwd, OSX_Intel reverse_tcp shell x86_64 and the Windows XP PRO SP3 - Full ROP calc shellcode as payloads. The particular results for each of the 18 baseline and additional benign datapoints experiments are shown in Appendix A. All the results obtained were done using a Gaussian random number generator for the coordinates and uniform random number of additional payload coordinates.

The overall results for the 18 data sets are summarized in Figure 14. The results are based on the centroids being classified as either malicious or non malicious for the purposes of activating the payload, based on lines 3-7 of the decoder shown in Algorithm 4. The focus of the results were concentrated on this part of the decoder, since it is the part that is most prone to failure and it is the one that depends on the actual K-means algorithm.

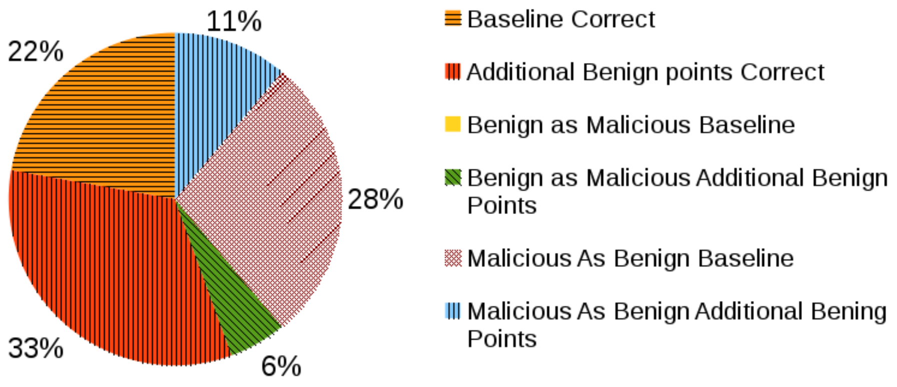


Figure 14. Summary of confusion matrix results for the K-means compromise

As can be seen from Figure 14, the addition of benign data points surpassed the baseline in terms of performance during the test runs carried out. A big percentage of the errors in the baseline came from not being able to activate the payload, due to having the centroid fall outside of the hull formed by the malicious payload data points.

The second part of the decoder, which obtains the hex numbers from the hull points is more resistant to failure and it does not rely on K-means itself. During the tests carried out, all trials successfully decoded the payload with a resolution of 4 decimal places. The results obtained can be explained by the fact that the decoder relies on the spread of the data on a torus described by a thin shell ϵ along a radius r and that the hull points inside the thin shell ϵ did not overlap with benign data points. The possibility of payload decoding failure once it is activated may be due to a mistaken data point that is classified as malicious in line 24 of Algorithm 4. This failure can be described as a benign point falling on the thin shell ϵ as shown in Figure 15.

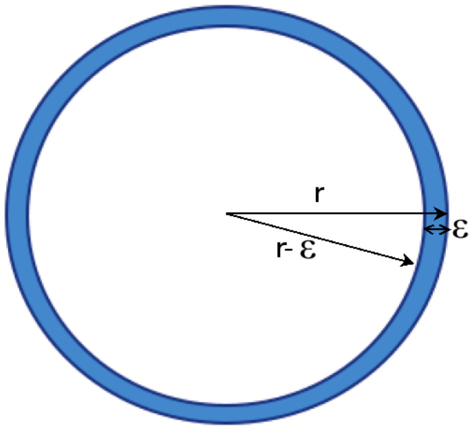


Figure 15. Volume of thin shell for the decoding string

The total area of the encoding region is given by the radius r of the two coordinates used in the description of the payload. The radius r is used to determine the inner boundary of the shell ϵ, whose length is determined by the decoding resolution of the implemented algorithm. As the decoder resolution increases in the decimal values needed to fall as an accurate representation of the radius, the ϵ shell decreases in volume and the more robust the algorithm becomes with respect to possible failure. That is to say, if the resolution of the decoder is just one decimal place, the length of ϵ is bigger than if the resolution of the decoder is 4 decimal places. Having a benign data point fall in the exact place with four decimal places is more difficult than having a benign point being misclassified by using just one decimal place.

The increased resistance to failure of the second part of the decoder is offset by the activation part of the payload. The results of Figure 14 can also be analyzed by the obtained DD values. A point of inquiry is whether the DS value from the selected sample during training correlates with the outcome of the test result. The DS values for the selected configuration against the DS obtained during the testing run is shown in Figure 16.

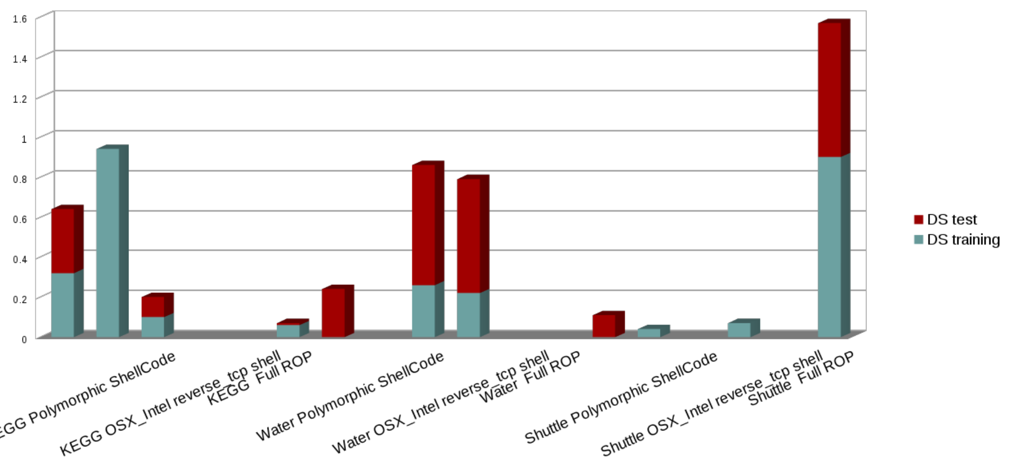


Figure 16. Comparison of DS value for selected configuration against the test run

The figure shows that there were only two instance where the DS for the training was zero; and for the test run, it was nonzero. Alternatively, there are three instances where the training DS was nonzero, and the test error is zero. The overall accuracy for the procedure is 33 percent error free. On the other hand, taking in as error only the instances where there was no detectable error in the training and error in the testing, gives an overall training selection accuracy of 72.2 percent. Based on this last finding, it was investigated if there were any correlation between training and testing outcomes. The r correlation coefficient was calculated to measure the degree of correlation between the two variables. The result indicated a moderate to strong relationship with a score of 0.795 score. While the measurement was relatively high no clear trend line was discernible from the gathered data that could provide further indication of correlation as shown in the Figure 17.

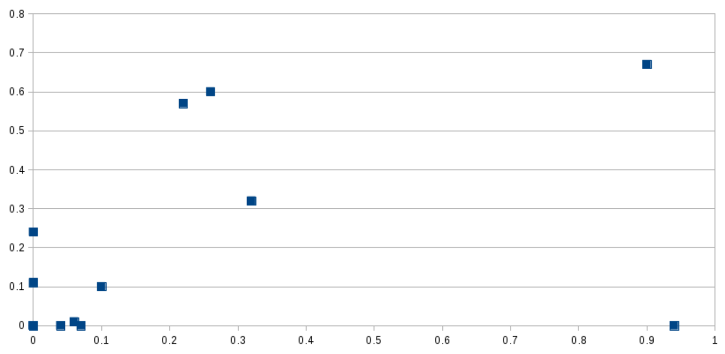


Figure 17. Plot of Training vs Testing DS of K-means Compromise

While the reported training selection accuracy was relatively high for the DS value where there was no detectable error in the training and error in the testing and strong correlation which is what an adversary is looking for, care must be taken with this interpretation. The first concern is that there was no clear trend line and the second is that the DS is an indicator of the overall confusion matrix error but it does not provide the type of error that occurred during the training and testing to see if the error types were the same. This is important, since there can be cases where the benign data cluster falls within the malicious boundary and may cause misclassification, but the malicious centroid fell within the boundary and depending on the attack composition, may have activated the payload. This type of error interpretation will depend on how the compromise is setup and if it is relevant to the activation of the attack itself.

Several factor can contribute to the results obtained in the training and test results. These include the specific data set used as well as the payload that was inserted into the data. Looking at the error in terms of the data sets that were used, it can be seen in Figure 18that the none of the data sets produced zero error in either their baseline or the additional benign datapoint insertion. While the Stalog Shuttle data set produced fewer errors it is not substantial to assert any statement on the relative performance advantage of the data set.

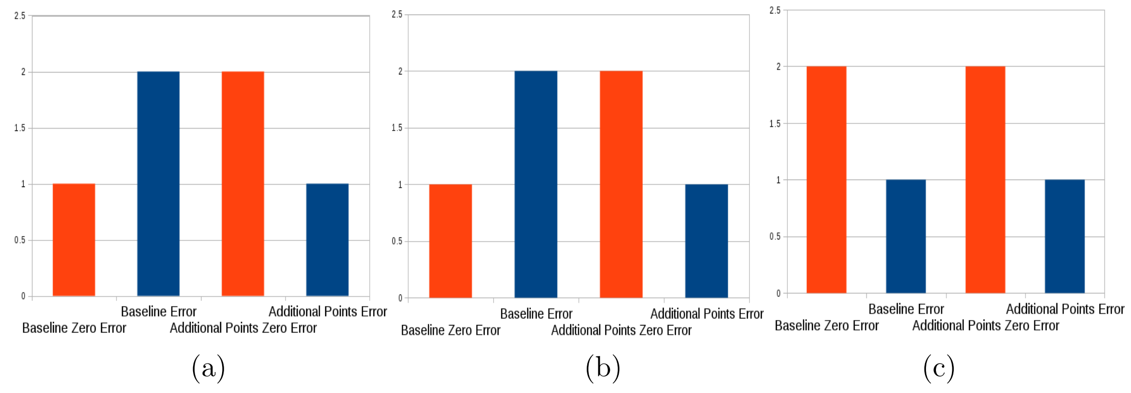


Figure 18. Errors for (a) KEGG Metabolic (b) Water Treatment (c) Stalog Shuttle data sets

The breakdown by data sets did not show significant differences in behavior, due to the sample sizes being too small to draw any definite conclusion, and should be further explored to determine their impact. These results contrast with some moderate differences that were found when considering the analysis by type of malware as shown in Figure 19.

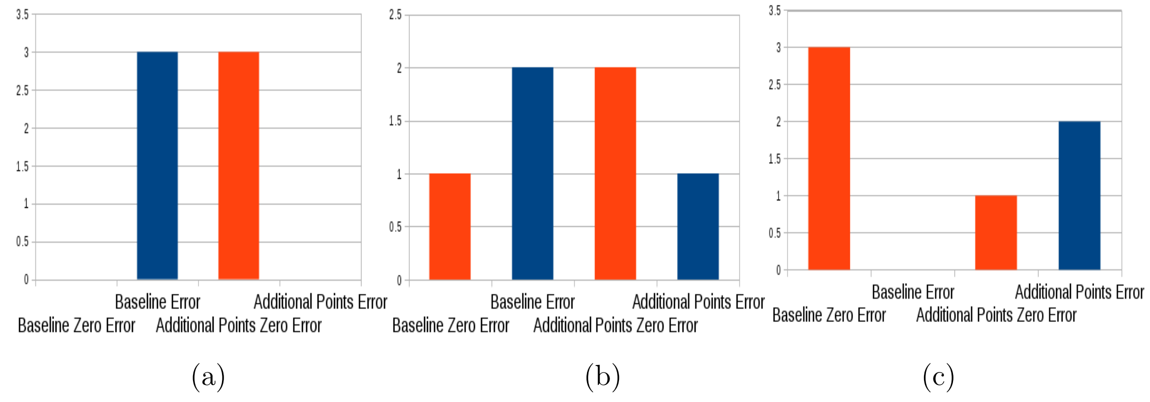


Figure 19. Errors from (a) Linux_x86 Polymorphic ShellCode (b) OSX_Intel reverse_tcp shell x86_64 (c) Windows XP PRO SP3 - Full ROP calc shellcode

The results for the Linux_x86 Polymorphic ShellCode showed that all attempt on the baseline configuration resulted in nonzero error. A possible explanation for this result is that this particular payload has the least amount of bytes of the three attacks selected, which implies less data points in the intended malicious cluster. The non baseline result for the Polymorphic ShellCode shows a performance difference in the three trial runs that were executed. The same can be said in the opposite case for the baseline error for the Windows XP PRO SP3, where the baseline error in all three instances was zero. Again, while these results are not conclusive, they yield some preliminary explanations on the behavior of the compromises.

The training set behavior yields useful information on the behavior of the compromises with respect to the data set by using the *DS* and the *DCD* as measures of compromise behavior. As shown in the Figure 20, as the *DCD* becomes higher, the *DS* becomes very low.

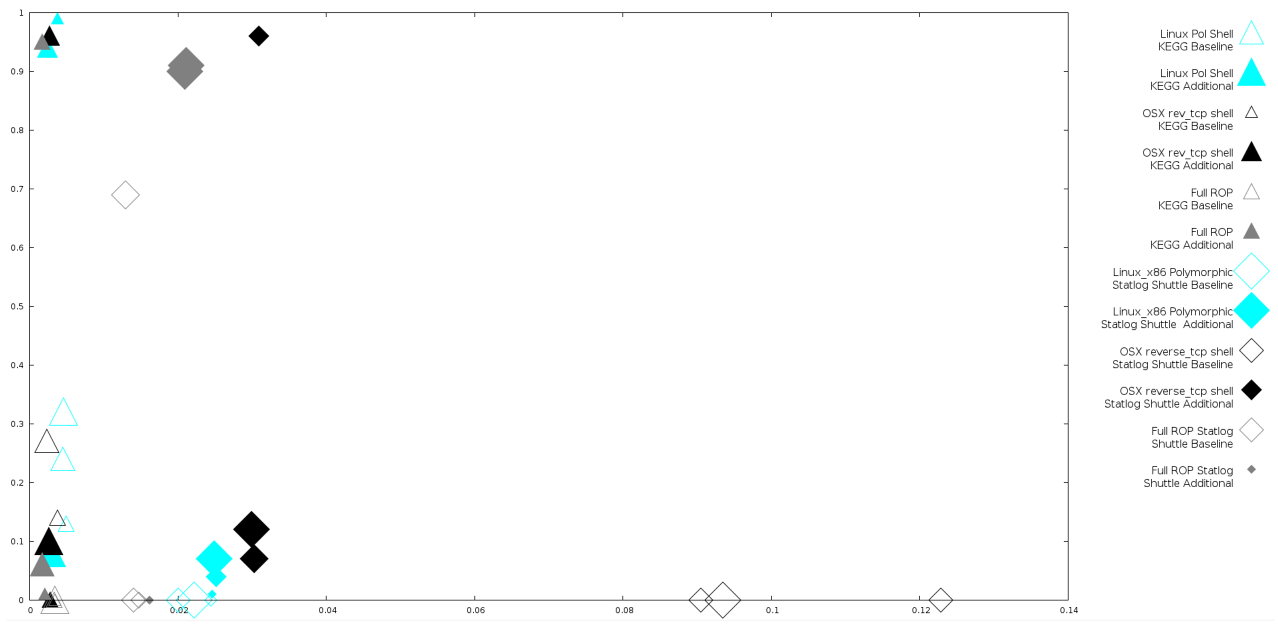


Figure 20. *DCD* and *DS* plot of the KEGG Metabolic and Shuttle data sets for the K-means compromise training. Size of symbols represent additional malicious elements added to move the clusters

This behavior was also seen in the Water Treatment aset showing a trend that is consistent with the theory that as the *DCD* becomes higher there should be a lower DS. Another interesting pattern is that due to the way the error was calculated, there is a center gap between 0.4 and 0.6. This pattern can be expected to vary depending on the error quantification strategy selected. The inclusion of additional payload elements is shown in Figure 20 represented by the size of the shapes in the figure. No discernible pattern was reflected in the executed runs carried out during the training. The figure also shows that the Shuttle data set has a bigger spread in values of the DCD with respect to the KEGG Metabolic data set. The water treatment had a bigger spread than both the KEGG and shuttle data set with respect to the DCD indicating that significant variability in the DCD can be expected among data sets.

An important point to take into consideration is the value of the DS during additional point insertion on the training phase. As shown in this example, from Table 25 of Appendix A, the *DS* measurement can give the wrong impression due to the number of additional elements added to simulate an additional benign data set.

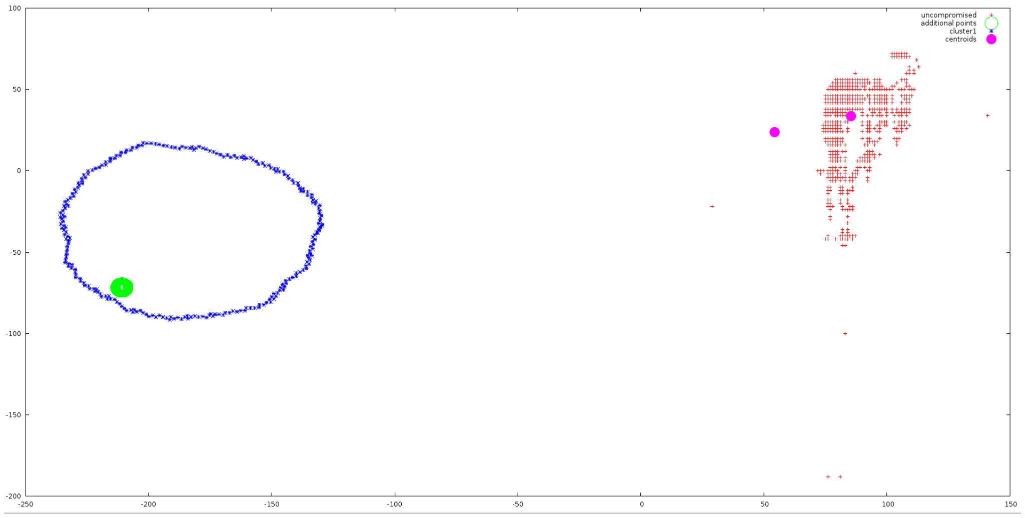


Figure 21. Example of possible failure of attack due to addition of points when the attack could be successful

Figure 21 shows the results of sample 2 that was selected as the best candidate to execute the attack. While the there is clear evidence that the additional malicious elements and the payload were not enough to drive the centroid to activate the attack during training, the same did not occur in the test scenario and could lead to overestimation of the number of additional malicious points to add during an attack. This overestimation may lead to higher degree of detection.

This section has covered some of the important elements of assessment in the execution of the K-means compromise carried out by an adversary. The following section will cover the results and interpretation of the test runs carried out for the Kohonen Neural Network compromise.

Experimental Results for Kohonen Neural Network Compromise

The Kohonen Neural Network implementation poses several constraints for the attacker in terms of fixed parameters which may make or break the viability of the attack, such as the size of the Kohonen layer and the data set size, among others. This section explores the Kohonen network in terms of a fixed Kohonen layer of 50 by 50 neurons, with a fixed constraint of 2,000 epochs and a learning rate of 0.005. The selection of the particular parameters were chosen to represent a small type of experimental setup which would be carried out during exploratory analysis of a data set before putting it into production. This type of scenario would be less susceptible to formal code verification. The parameter constraints given are usually fixed by the person running the experiments and would fall outside of the attackers' control. This would leave the attacker with the selection of additional data samples to inject into the application as the only free parameter in an effort to try to shift the training in favor of the payload execution. The size of the experiment will be limited to 700 randomly chosen samples from the selected data sets. From these, it is assumed that the attacker has intercepted 200 samples from the data set which are excluded from the testing set.

The three simulations made by the attacker were structured using three different positions for the payload within the benign data set. The locations were randomly selected to simulate various possible conditions under which the attack may be carried out. Under real training conditions, the attacker would select the position which reflects a distinct advantage during a compromise to maximize his possibilities of success.

The results of the tests carried out for the Kohonen network compromise documented in Appendix B are shown in Figure 22. As with the K-means algorithm, these results reflect the intrinsic error and are not based on the decoder resolution. The results show the outcome of the integer value discrimination in the decoder algorithm 7. An interesting result is that zero error was not achieved in any of the runs using the default evaluation criteria for the Kohonen algorithm. Notwithstanding, the payload was decoded successfully in every instance using four decimal places in the extract value function of the decoder sequence of the algorithm. The selection of the decoding resolution will vary depending on the additional amount of code that can be inserted into the compromised program, to dictate the final metric to be used in actual compromises.

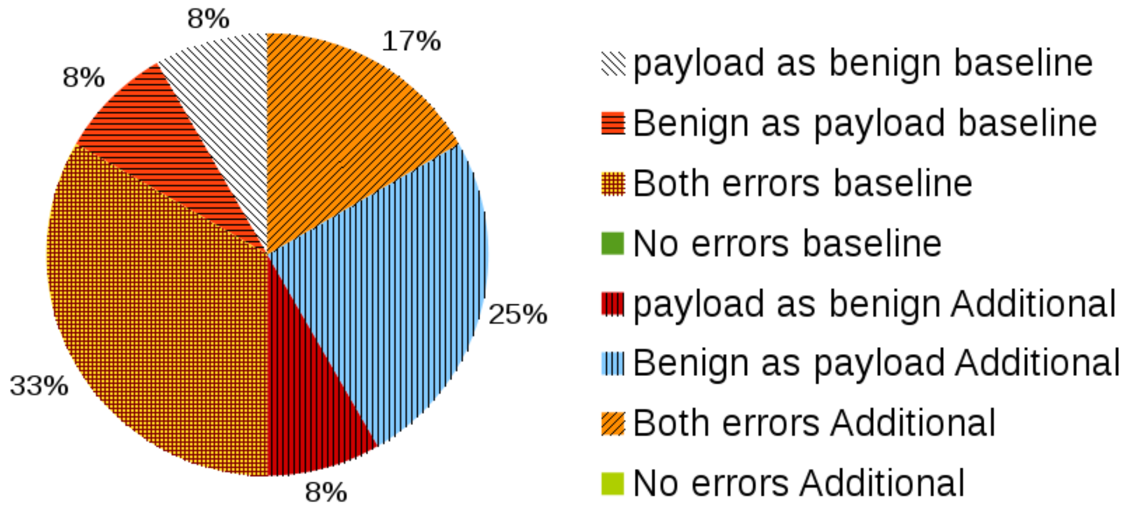


Figure 22. Summary of confusion matrix results for the Kohonen compromise

Figure 23 shows that there is no clear trend line between the training versus testing *DS,* while the Pearson coefficient shows a weak correlation at .55. The lack of a trend line is also indicative that there is no clear indication of relationship between the training *DS* and testing *DS.*

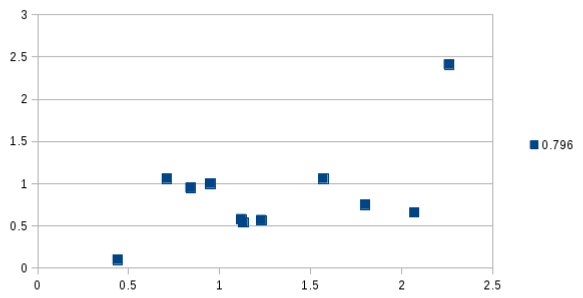


Figure 23. Plot of training vs testing DS of the Kohonen compromise

The training of the Kohonen network's DS and DCD readings are shown in Figure 24. The output shows that while the the empty upper right quadrant is smaller than K-means algorithm, it still follows the intuition of the proposed formulas. Some of the possible reasons for this deviation may be attributed to the particulars of the encoding and specific data sets chosen, in addition to the Kohonen's lateral feedback. While the K-means algorithm treats each cluster independently, the Kohonen network's lateral feedback influences nodes that are in the vicinity of each other.

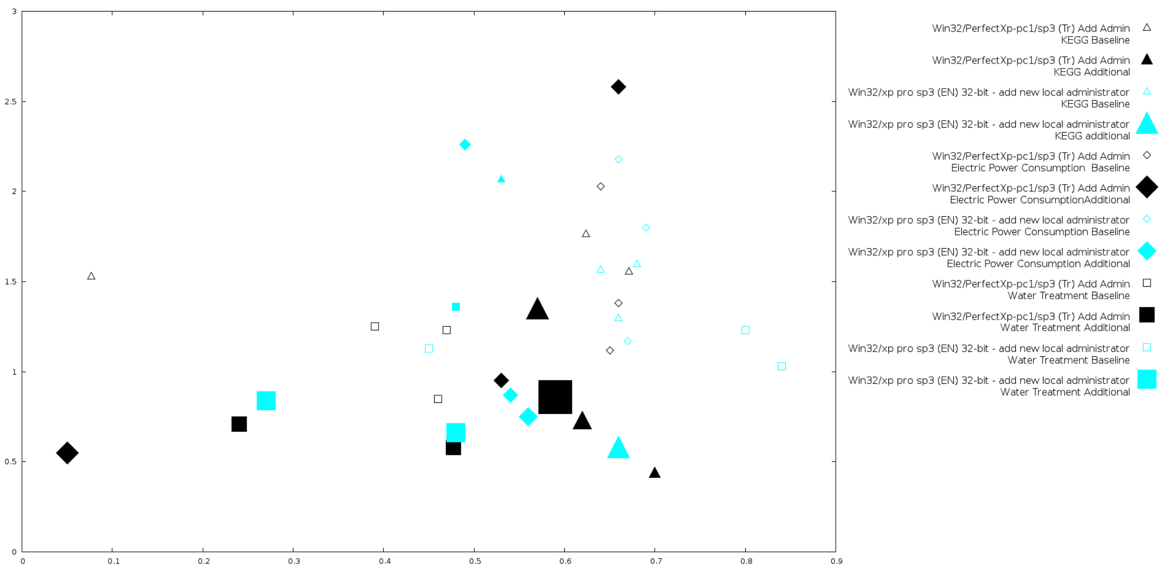


Figure 24. DCD and DS plot of the three data sets for the Kohonen compromise training. Size of symbols represent additional malicious elements added to move the clusters

This phenomena may initially pull neighboring nodes closer to each other, and in some particular cases, may contribute to additional convergence error and therefore a shift in the *DS* towards centered *DCD* values.

In addition to the difficulty of convergence, the addition of malicious payload to achieve convergence needs to be considered during training. The figure also shows the results of the *DCD* and the *DS*, while additional payload elements are represented by the sizes of the icons. The impact of the error minimization of additional malicious payload data points can also be seen in the epochs as the neural network is trained.

This impact can be seen in Table 29 of the Appendix B, where sample 1 is shown with an additional payload sequence. This means that to achieve zero error, the attacker had to reinsert the 112 vector sequence once more to achieve the desired result. In summary, for that particular sample, the training entails: 140 intercepted vectors, 112 samples inserted twice, and 505 randomly generated samples, for total 869 samples of training data. The figure 25 shows the training set, with the top line being the single insertion of the payload, and the bottom one is the payload that was inserted twice, that achieves zero error

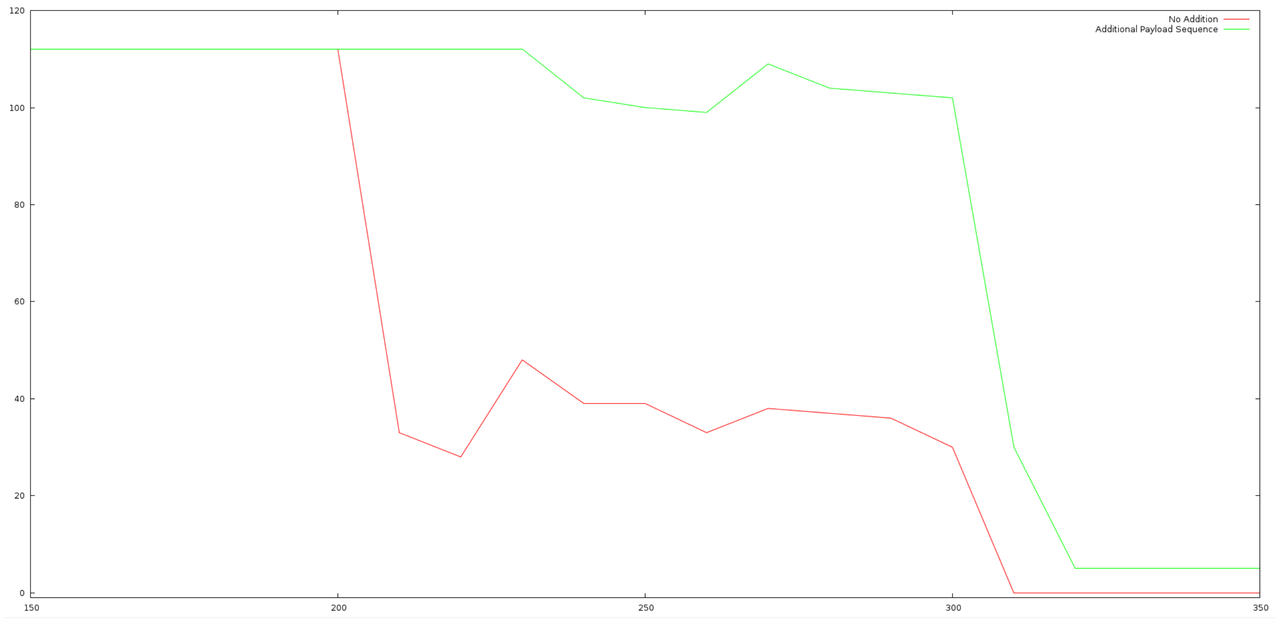


Figure 25. Comparison of insertion of additional payload sequence and single insertion of payload sequence

Additional Results for Kohonen Neural Network with Different Kohonen Layer Sizes

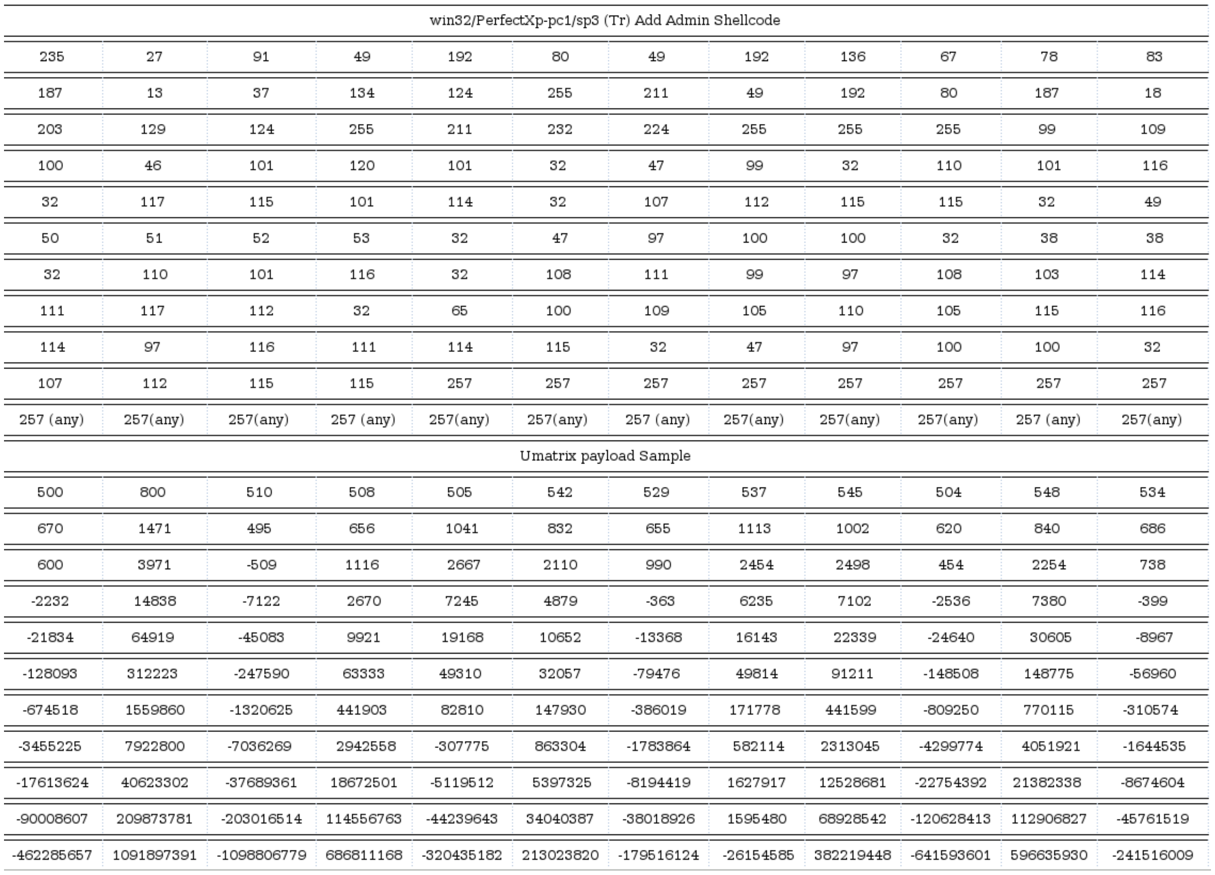
The Kohonen experiments were done with a 50*50 fixed layer size. An important question to answer is what effect on the payload execution does to the variation of the Kohonen layer size have when the end user that utilizes the compromised algorithm. The data set used for this test was the KEGG Metabolic Relation Network Data Set Baseline using win32/xp pro sp3 (EN) 32-bit - add new local administrator payload. 22 offsets were selected for the order column offset, and 31 for the payload column offset. The epochs and the learning rate used were the same as those for the rest of the Kohonen experiments. The results obtained for the different Kohonen layer sizes are shown in Table 4. As can be seen from the table results, the same caution as with the K-means algorithm needs to be taken with the Kohonen algorithm's *DS* reading. While the *DS* reading can be a good confusion matrix summary, it does not necessarily reflect the type of error that the algorithm is obtaining due to the data. The table shows that while the Kohonen layer size grows, so does the *DS* value. While this may not be indicative of a definitive trend, it does highlight a possible point to consider on the part of the adversary when deploying the attack.

Size	*DS*	Error in Payload
50*50	1.0600	2
40*40	0.6712	7
30*30	0.4149	57
20*20	0.1494	94

Table 4: Results of varying Kohonen layer sizes using the KEGG Metabolic Relation Network Data Set Baseline and win32/xp pro sp3 (EN) 32-bit - add new local administrator payload

Experimental Results for Kohonen Neural Network U-Matrix Attack Variant

The Kohonen Neural Network U-Matrix compromise merits a separate discussion from the previous Kohonen compromise. While the compromised algorithm is the same, the characteristics of the encoding do change and merit closer inspection, since they will impact the results obtained. As discussed in the previous chapter, the encoding of the U-Matrix compromise is an actual embedding of a submatrix within the matrix that comprises the Kohonen layer. A common observed characteristic shared by all the payloads encoded was that there was a substantial spread in the magnitude of the numbers embedded as part of the compromise. This phenomenon can be seen in Figure 26. This will have an influence in the Kohonen algorithm with its convergence results, especially with lateral feedback.



win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode											
235	27	91	49	192	80	49	192	136	67	78	83
187	13	37	134	124	255	211	49	192	80	187	18
203	129	124	255	211	232	224	255	255	255	99	109
100	46	101	120	101	32	47	99	32	110	101	116
32	117	115	101	114	32	107	112	115	115	32	49
50	51	52	53	32	47	97	100	100	32	38	38
32	110	101	116	32	108	111	99	97	108	103	114
111	117	112	32	65	100	109	105	110	105	115	116
114	97	116	111	114	115	32	47	97	100	100	32
107	112	115	115	257	257	257	257	257	257	257	257
257 (any)	257(any)	257(any)	257 (any)	257(any)	257(any)	257 (any)	257(any)	257(any)	257(any)	257 (any)	257(any)
Umatrix payload Sample											
500	800	510	508	505	542	529	537	545	504	548	534
670	1471	495	656	1041	832	655	1113	1002	620	840	686
600	3971	-509	1116	2667	2110	990	2454	2498	454	2254	738
-2232	14838	-7122	2670	7245	4879	-363	6235	7102	-2536	7380	-399
-21834	64919	-45083	9921	19168	10652	-13368	16143	22339	-24640	30605	-8967
-128093	312223	-247590	63333	49310	32057	-79476	49814	91211	-148508	148775	-56960
-674518	1559860	-1320625	441903	82810	147930	-386019	171778	441599	-809250	770115	-310574
-3455225	7922800	-7036269	2942558	-307775	863304	-1783864	582114	2313045	-4299774	4051921	-1644535
-17613624	40623302	-37689361	18672501	-5119512	5397325	-8194419	1627917	12528681	-22754392	21382338	-8674604
-90008607	209873781	-203016514	114556763	-44239643	34040387	-38018926	1595480	68928542	-120628413	112906827	-45761519
-462285657	1091897391	-1098806779	686811168	-320435182	213023820	-179516124	-26154585	382219448	-641593601	596635930	-241516009

Figure 26. win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode with its Corresponding U-Matrix Encoding

The results of the tests carried out for this compromise in Appendix C are summarized in Figure 27. As can be seen in the comparison between both the Kohonen and the Kohonen U-Matrix compromise, the results vary substantially in composition. The biggest error in the Kohonen U-Matrix compromise is the benign data considered as payload in the test carried out, by adding additional benign samples to the data set. This contrasts with the Kohonen's previous compromise, in which the biggest error was in the baseline, and consisted in both types of error, benign as payload and payload as benign.

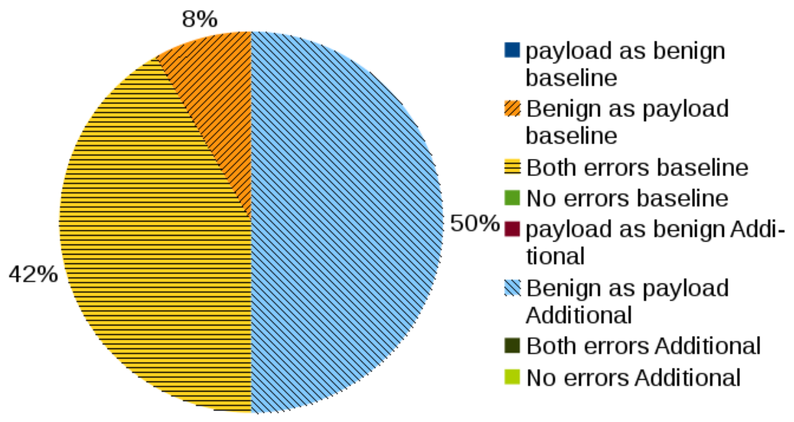


Figure 27. Summary of confusion matrix results for the Kohonen algorithm U-Matrix compromise

To further investigate the results obtained, a correlation analysis was made, as with the previous algorithms. A weak correlation of 0.59 was obtained. The plot of the training vs testing *DS* is shown in Figure 28, where it can be seen that there is a slight trend, but it is not of a linear nature.

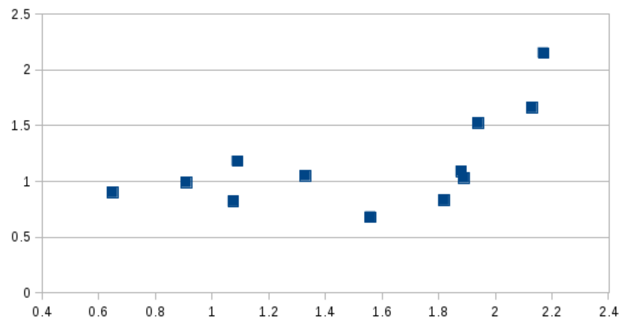


Figure 28. Plot of training vs testing *DS* of Kohonen U-Matrix compromise

The plot of the *DCD* versus *DS* sustains the results observed with the previous algorithms, in that as *DCD* becomes higher, there should be a lower *DS*.

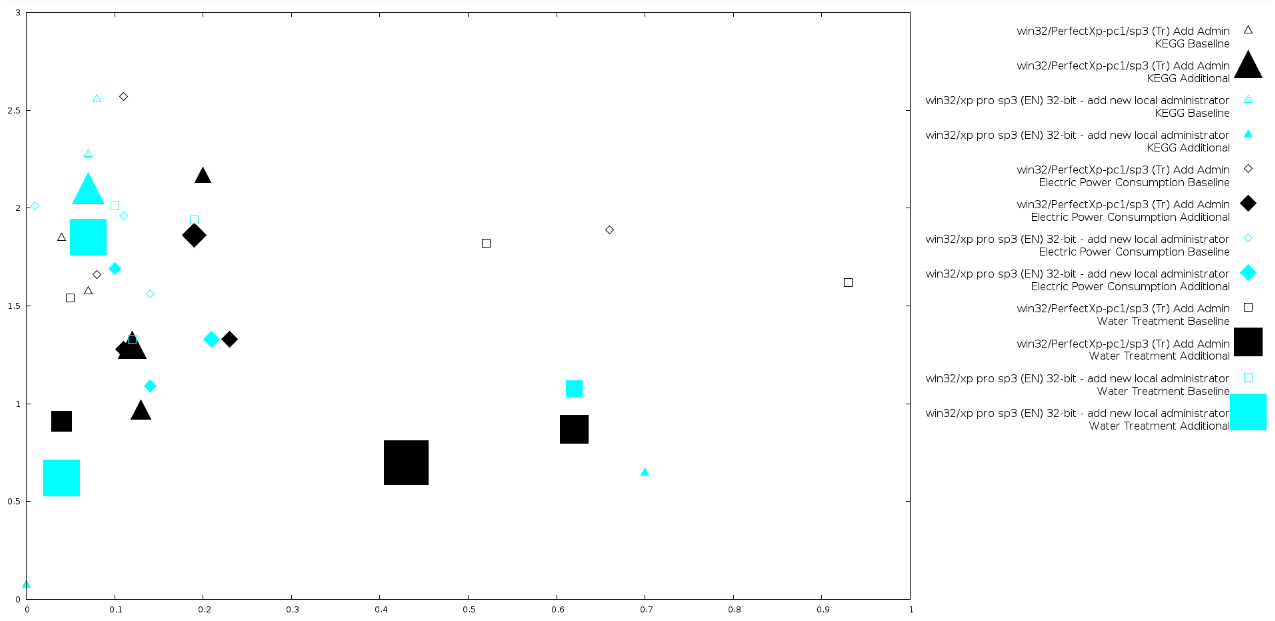


Figure 29. *DCD* and *DS* plot of the three data sets for the Kohonen U-Matrix compromise training. Size of symbols represent additional malicious elements added to move the clusters

An additional consideration to take into account for this compromise is the padding for the compromise. As discussed in the previous chapter, the U-Matrix compromise needs a padding at the lower part of the matrix to be feasible. It was found that on Table 44, in the confusion matrix, payload misclassified as benign was composed of the padding points, and not on the payload attack points themselves. This points to caution on the part of the adversary, since the selection of the padding values may also be critical in establishing zero error.

Experimental Results for Adaptive Resonance Algorithm

The Adaptive Resonance 1 Algorithm uses binary input; and outputs a cluster assignment based on a cluster prototype. In this work, all computations were made by converting the binary values back to decimal to maintain uniformity in the measurements with regards to other algorithms. To carry out the tests described in the previous chapter for Adaptive Resonance 1, all inputs from the data sets that were not of integer value were converted by rounding to the closest integer, using a fixed learning rate of 2.0. All attacks were carried out by using zero values in the rest of the vector that comprises the payload. The selection of the zero value as the padding for the Adaptive Resonance was due to the way the algorithm classifies the data. The vigilance parameter determines the granularity of the clusters based on the differences between binary vectors. If the vectors were filled with random values, their binary equivalent would add too much noise to the data processed by the algorithm. The drawback of this technique is that while it increases the probability of successful attack, it can also lead to easier detection.

The test results of Appendix D are shown in Figure 30.The results show that, as opposed to other algorithms, the Adaptive Resonance response to payload injection is very favorable due to its high success rate. Nonetheless, the encoding conversion to binary values may be enough to arise suspicion since binary vectors will appear unreasonable large if done on a data set with small values.

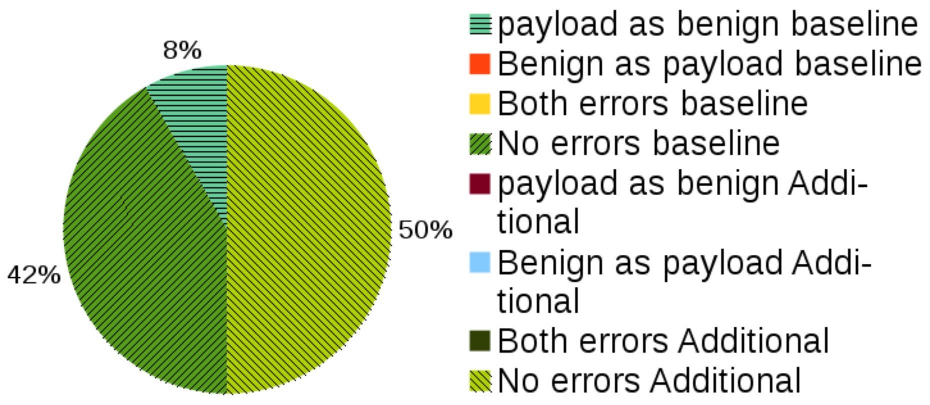


Figure 30. Summary of confusion matrix results for the Adaptive Resonance algorithm compromise

Figure 31 demonstrates a graph of the vigilance parameters used in the test results displayed in order of increasing value. The figure demonstrates that zero error is achieved while the vigilance parameter falls below a given vigilance value.

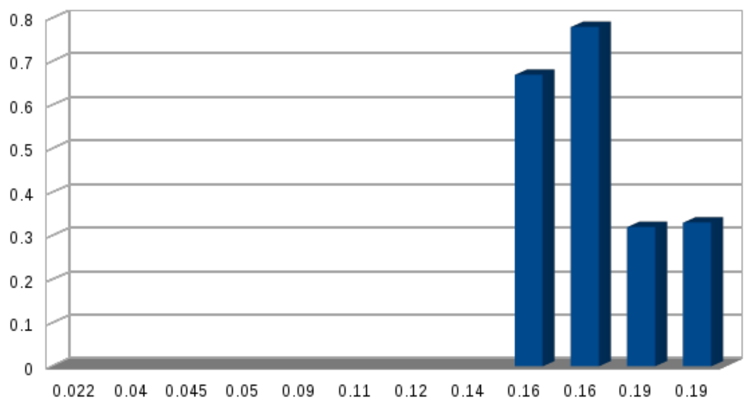


Figure 31. Sorted vigilance parameter on the x-axis with its corresponding *DS* value result for the test runs

While the algorithm displays several favorable qualities, such as encoding simplicity and a high degree of success versus other algorithms, it has some drawbacks in terms of analyzing the training of the algorithm. Being binary valued in nature, the algorithm's success was hard limited in terms of being successful or not. The densities remained the same throughout the training, since the cluster assignment into either malicious or benign is independent of the number of clusters. Another limitation is that while the *DS* is a good indicator of success, it was dependent mainly on the value of the vigilance parameter that falls outside of the attacker's control. While there might be perfect execution of the algorithm under training, it may completely fail if the vigilance parameter varies considerably. This phenomenon occurred only in a limited number of tests, however it remains a considerable factor when the adversary is implementing the algorithm.

Another result that was obtained, which might be interpreted as a drawback, is the example in Table 5, which shows an interesting behavior in the *DCD*. These results were derived from Table 64 of Appendix E. The results illustrate that the *DCD* must be carefully handled for unsupervised algorithms that do not specify in advance the number of clusters, since the cluster distance may be warped by the equation's denominator as the number of clusters grow.

Sample 1 with *DCD* of 0.180					
	Trojan Payload		Benign Data		
Cluster	0		1	2	3
Frequency:	51		130	1491	1279

Sample 2 with DCD of 0.210					
	Trojan Payload		Benign Data		
Cluster	0	1	2	3	4
Frequency:	24	27	521	1277	1102

Sample 3 with DCD of 0.074			
	Trojan Payload		Benign Data
Cluster	0	1	2
Frequency:	31	20	2900

Table 5: Results of Statlog (Shuttle) Data Set Baseline using Windows XP PRO SP3 - Full ROP calc shellcode

Effects of Varying the Number of Data Columns and the Vigilance Parameter

The results demonstrated for the tests made on the Adaptive Resonance algorithm in the last section were done using a fixed number of columns and a fixed vigilance parameter throughout all runs. This section covers the impact of the selection of columns and vigilance parameter in a given data set and the implications for the attacker's design of the compromise.

The tests were started by varying the number of columns using a fixed vigilance parameter of 0.3 and a fixed learning rate 2.0. The tests were carried out on the KEGG Metabolic Network Data Set by selecting 10,683 random instances from the data set using OSX/Intel - setuid shell x86_64 payload. The ranges used for the payload and benign data set tested are shown in Table 6. The addition of columns were also filled with zero vectors to limit the amount of influence due to random variations in non payload columns.

Data Set			Payload 1		Payload 2	
	Min	Max	Min	Max	Min	Max
Column 1	2	116	170	225	257	312
Column 2	1	509	1024	1279	2049	2304

Table 6: Minimum and maximum of data set range of the two columns used along with the Payload Range

Figure 32 shows the output for payload 2 and benign data. The graph starts by showing the results of just three columns from the data set and incrementally showing the addition of more columns from the data set to make a larger feature vector. While there is a clear trend towards reduction of clusters as the columns from the data set are added, (showing signs of homogeneity as the addition progresses), a sudden change occurs in column 5, indicating that the pattern of homogenization depends on the column values. In this case, column 5 of the data set has a significantly smaller range than the rest of the first eight columns sampled.

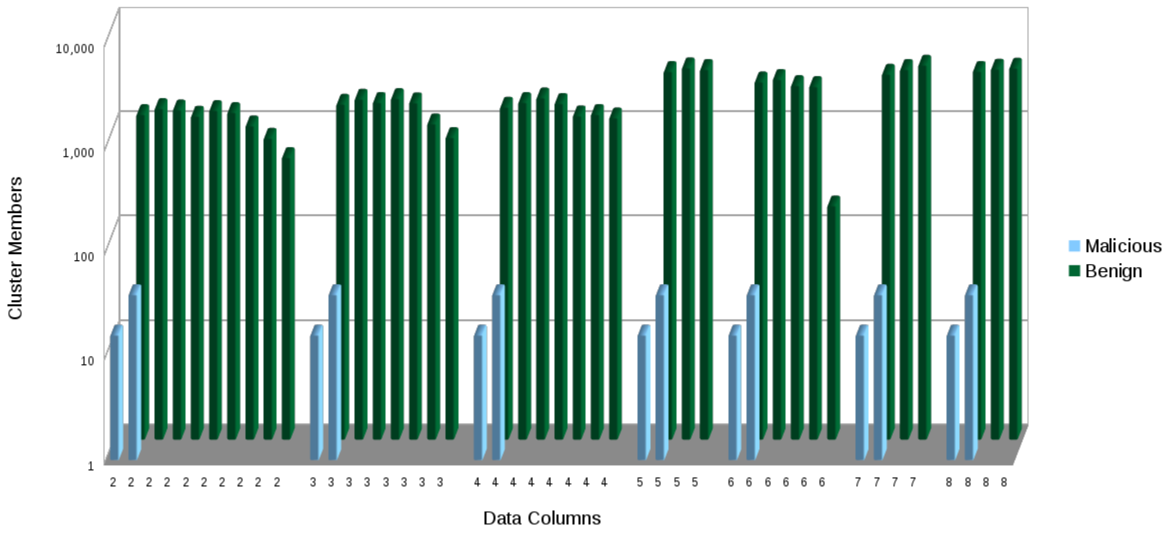


Figure 32. Log plot showing results of data set column additions for payload 3 of Table 6

Table 7 shows the average values of DY_{Trojan} DS and DCD for the clusters obtained for each run. As it should be expected, the DS will be decreased as the number of elements belonging to cluster 1 increases, with the addition of columns making the denominator larger in magnitude. The density for the malicious cluster remained fixed during the duration of the addition of the columns.

The reason behind the fixed density is that the elements inside the clusters and the dimensions of the column grew at a constant ratio, therefore giving the same result. The DCD value behavior can be explained by observing that the number of cluster decreases as the number of rows increases. The row increase allows for greater homogeneity on the ratio of zeros to ones in the encoding, giving way to a consolidation of clusters. This will mean that the numerator in the DCD computation will decrease in magnitude. While the distances of the centroids may vary, the decreasing number of clusters is enough to influence the computation of the DCD towards smaller values.

	Payload 2		
Columns	DS	DY_Trojan	DCD
2	0.0287	0.0011	2.64
3	0.0230	0.0011	2.34
4	0.0245	0.0011	2.33
5	0.0113	0.0011	1.55
6	0.0142	0.0011	1.19
7	0.0120	0.0011	0.92
8	0.0112	0.0011	0.92

Table 7: Impact of column additions on DY_Mal DS and DCD

Figure 33 shows the vigilance parameter effects on the same data set using payload 1. The results show how the clustering arrangement for the payload and how the malicious payload achieves total exclusion of the benign sets as the vigilance parameter is lowered. The lowering of the vigilance parameter makes the cluster threshold bound tighter, and therefore creates a more crisp segregation between the payload and the benign data.

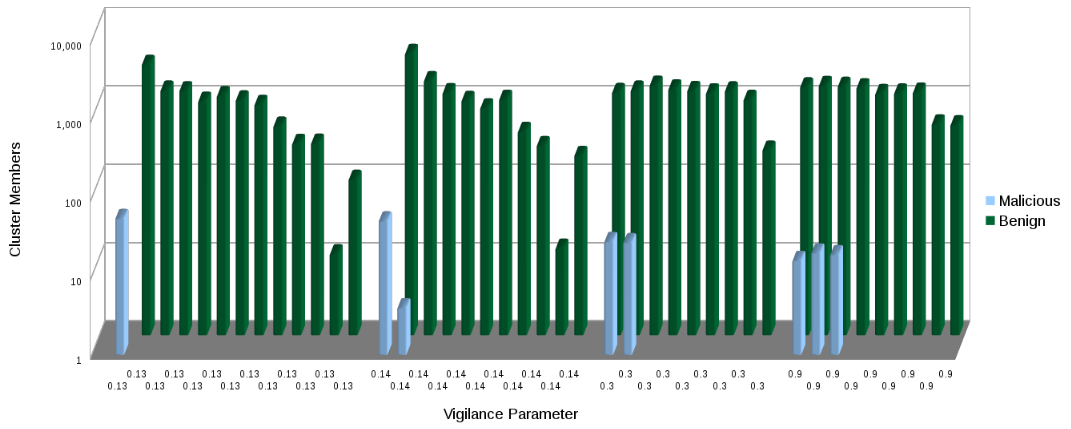


Figure 33. Log plot of vigilance parameter behavior

For the payload 1, the total segregation happened at the vigilance parameter of 0.13, while the payload 2 segregation occurred at 0.16. This places a constraint on the attacker, since the vigilance parameter has to be estimated in order to carry out a successful attack.

Table 8 shows the results of the DS, DY_{Trojan}, and DCD, as the vigilance parameter is increased in payload 1. Zero error was achieved in this instance at a value between the 0.13 and 0.14. This is the same response as payload 2 but with a different threshold value for the vigilance parameter, and zero error in classifying the payload from the benign data.

	Payload 1		
Vigilance Parameter	*DS*	DY_Trojan	*DCD*
0.13	0.0000	0.0038	3.02
0.14	0.0727	0.0036	2.74
0.30	1.0000	0.0000	0.00
0.90	1.0000	0.0000	0.00

Table 8: Parameters *DY, DS,* and *DCD* for the vigilance parameter variation

Significance Tests on the Use of a Particular Algorithm Attack on Shared Data Set

The use of the same data set on different machine learning algorithms lead to the question on whether compromising a particular algorithm and using a particular data set is more advantageous. To carry out this analysis, the Friedman nonparametric test was utilized. The results analyzed were based on the error obtained during the test executed, and were normalized using the total number of samples in the test run. The algorithm that obtained the least normalized error was chosen to be the biggest score.

The selected data set, which overlaps all algorithms, is the KEGG Metabolic Network Data Set Using Baseline and the additional benign data point addition tests. For this data set, the number of tests carried out on simple Kohonen and Kohonen U-Matrix compromises were less than those for the K-means and Adaptive Resonance compromises. To resolve this difference, two more tests were added to both Appendix B and C to match the number of samples for the KEGG Metabolic Network Data Set to three runs baseline and three runs with additional benign data sets. This allows to approximate the Friedman rank test F to the χ^2 distribution with *k-1* degrees of freedom (Stamatis, 2002). The formula for this statistic is:

$$F = \frac{12}{Nk(k+1)}\left[\sum R_i^2\right] - 3N(k+1) \tag{39}$$

For the Friedman test, it was assumed that:

H_0: There is no difference among learning algorithms used to insert the payload.

H_a: There is a difference among learning algorithms used to insert the payload.

The scoring for the Friedman test is presented in Table 9.

The result for the Equation 39 was:

$$F = \frac{12}{6*4\,(4+1)}\left(18.5^2 + 10^2 + 9^2 + 22.5^2\right) - 3*6\,(4+1) = 11.125 \tag{40}$$

The value of $\chi^2_{.005} = 12.838$, thus the result is that $11.125 \not> 12.838$, giving way to a rejection of the alternate hypothesis and accepting that the null hypothesis is correct.

	K-means	Kohonen	Kohonen U-Matrix	Adaptive Resonance	
	2.000	3.000	1.0	4.00	
	3.000	1.000	2.0	4.00	
	3.500	1.000	2.0	3.50	
	3.500	2.000	1.0	3.50	
	3.500	2.000	1.0	3.50	
	3.000	1.000	2.0	4.00	Column Sum
Sums	18.500	10.000	9.0	22.50	40
Observed Means	3.083	1.667	1.5	3.75	10
Null Hypothesis	2.500	2.500	2.5	2.50	10
Sample Count	6.000	6.000	6.0	6.00	24

Table 9: Results of Friedman test scoring for the KEGG Metabolic Network data set with baseline and additional data points

The graph of the mean rank of the expected null hypothesis versus the obtained mean ranks appears in Figure 34:

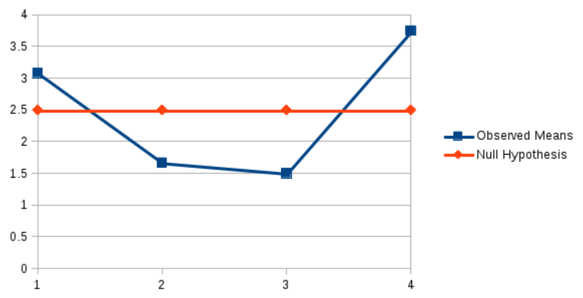


Figure 34. Friedman values for the mean rank of Expected Null Hypothesis vs observed

Significance Tests of Baseline Versus Additional Samples in Data Set

An important part of the experimental design presented throughout this dissertation is the tests between a baseline of intercepted messages and the addition of generated benign

Baseline	Additional	Signed Rank
89	100	5
97	100	3
100	97	-3
62	100	6
59	60	1
100	100	0
97	100	3
100	100	0
100	11	-7

Figure 35. Results of Wilcoxon signed ranked test for the K-means compromise

samples. The addition of samples is intended to simulate the rest of the data that was not intercepted by the attacker, in an effort to make the training more realistic with regards to the eventual size of the analyzed data. To achieve this comparison, all test results were normalized using the error over the total number of samples. This result was then subtracted from one to obtain a normalized score.

To answer the question whether the tests with added samples are effective, a Wilcoxon significance test was carried out for each of the algorithms tested. For the test, it is assumed that:

H_0: The distribution of baseline intercepted samples and additional benign samples is the same.

H_a: The distribution of the additional generated samples varies substantially from the baseline.

For the K-means algorithm, the signed ranked score is given in Table 35.

The rank-sum test statistic W for the K-means sample is 10, with the number of samples $n = 7$. The critical W for 5% level is 2. Therefore, the results demonstrate that the null hypothesis must be accepted, since $W >= \textit{critical } W$.

The Kohonen compromise test results are given in Table 36

Baseline	Additional	Signed Rank
96	97	1.5
84	92	5
93	95	3.5
93	94	1.5
97	97	0
94	92	-3.5

Figure 36. Results of Wilcoxon signed ranked test for the Kohonen compromise

Baseline	Additional	Signed Rank
85	86	1
90	83	-4
86	95	5
94	90	-3
87	85	-2
89	89	0

Figure 37. Results of Wilcoxon signed ranked test for the Kohonen U-Matrix compromise

In this case, the obtained W value was 3.5. The critical value of W for $N = 5$ at $p \leq 0.05$ is 0. Therefore, the result is not significant at $p \leq 0.05$. Under this result, the null hypothesis must also be accepted for the Kohonen test results.

Finally, the results for the Kohonen U-Matrix test results are given in Table 37, where $W = 6$, and the critical value of W for $N = 5$ at $p \leq 0.05$ is 0. Therefore, the result is not significant at p $\leq$ 0.05.

Adaptive Resonance was not included, since the errors obtained in the tests were less than 0.5, giving zero difference throughout all samples, when rounded to a percentage number without decimal positions in all the Wilcoxon Signed Rank results.

Performance Analysis of the Compromises

The results presented where based on an evaluation that included two factors that decreased the accuracy results. The first factor is that the criteria contemplated both the stealth and the accuracy of the compromise. Both of these goals an be visualized as opposing objectives in a gradient as shown in Figure 38. The second factor was the random selection of the measurements that added to the decrease in accuracy of the results obtained.

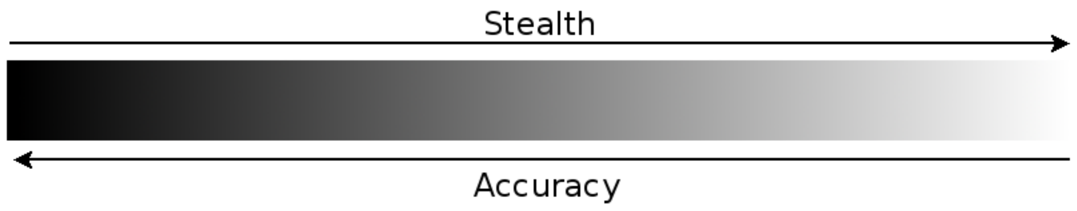


Figure 38. Compromise vs. stealth gradient

It can be argumented that whenever the adversary chooses this tradeoff of stealth versus accuracy the attack results will be mixed as those obtained in the experiments carried out in this dissertation. In the next paragraphs a brief explanation using K-means, the Self Organizing Feature Map, and Adaptive Resonance algorithms will be given as examples of why this tradeoff is intrinsic of the nature of the attack. Empirical tests of the analysis is proposed as future work.

The K-means algorithm was presented in chapter three and it described the steps carried out by the algorithm. In the algorithm execution, the second step is to assign the sample points according to the distance metric comparison of: $|x - \mu_j| < |x - \mu_k|$. Distance measures for segregation assume that if the clusters are separated by enough distance then discrimination into the correct cluster can be done. This is the first assumption that must be sacrificed under a K-means attack (and under any distance measurement unsupervised algorithm) if stealth is desired. For the attacker to be effective in providing stealth the payload must be close to the real data as possible. If this is not done, the payload may appear as unreasonable outliers that can alert the end user that the data is wrong, and may lead to suspicion of an attack taking place. Under the tests carried out, the locations were randomly chosen which violated optimal placement that increases the accuracy of the results but gave reasonable stealth to the attack.

The second point to consider in the tradeoff is the result of step 3 of the algorithm that calculates the new centroids by using the update rule: $\mu_k = \frac{1}{N_k}\Sigma_{q=1}^{N_k} x_q$. The update rule is sensitive to two specific issues. The first is the presence of an outlier within the elements contemplated within the assigned cluster. The second is the number of elements that are part of the assigned cluster. Both of this issues can be controlled by the adversary to the desired degree of accuracy. By fulfilling the distance criteria (given by the distance measure of step 2 of the algorithm) the attacker can then choose the appropriate encoding to minimize the possibility of an outlier being present in the dataset. This will guarantee that the numerator will be roughly homogeneous within the update rule and will not bias the mean measurement of the centroid. The second issue is the number of elements that are assigned to the cluster that, if the attacker fulfills the distance criteria, then he can inject additional instances of the attack or closely spaced points to bias the centroid even more towards the attack surface.

For the Self Organizing Feature Map The distance is calculated using: $Distance_j = \sum_{i=1}^{n} (Input_i - W_{old\,ij})$ which offers the same tradeoff for the adversary. Nonetheless, there are three major differences in carrying out the SOFM compromise that need to be carefully analyzed. The first one is that individual payload elements are assigned to individual neurons as opposed to clusters in K-means. Individual assignment of payload elements will depend on whether a node can be assigned on a one to one correspondence to the payload value. This has to be forecasted by the adversary using as guide the results obtained in the experimental section on different Kohonen layer sizes. As the Kohonen layer sizes decrease an additional number of payload copies have to be injected by the adversary to close the distance within epoch calculations and decrease stealthiness. Notwithstanding, the distance can be arbitrarily small by the attacker by injecting enough payload copies given a reasonable size Kohonen layer which can support the mixture of payload and benign data.

The second difference lies in the lateral feedback impact on payload elements with benign neighbors. The impact of the lateral feedback imposes another hurdle that can be remedied by increasing the number of payload copies into the algorithm to compensate for the lateral feedback "pull" from neighbors on payload nodes. Additional copies of the nodes have to be injected by the adversary to negate the effects of the pull of the neighbors on the final payload value with the same result of reducing the stealth of the attack.

The final difference lies in the weight adjustment by the learning coefficient c and the neighborhood function η of equation 23. These two parameters will impact the within epoch learning rate and convergence of the neural network to the desired values. The setting of these values will definitely determine the convergence of the nodes upon the payload values and this must be compensated by the adversary by additional copies to be calculated during a particular epoch of the algorithm. This third parameter will also increase the number of copies if the parameters are to low and make the learning steps slow in convergence.

The Adaptive Resonance Algorithm works in a winner take all fashion in which the closeness to the clusters is already formed by the algorithm. The closeness measure is given by equation 37 which is governed by the vigilance parameter. If the left hand side of the equation falls below the established vigilance parameter it rejects the cluster assignment until all clusters are compared and a new cluster is formed. The success of the compromise depends mostly on the previous consideration which takes into account the following two elements. The payload values should be close enough in value such that the conversion to binary values should remain below the vigilance parameter. This implies careful selection of the exploit to cluster the values close enough within a single cluster. The second element to consider is that the separation of binary differences between the payload and the benign data set are enough so that benign data fails the vigilance test and fall in a cluster that is not part of the payload. These criteria set the theoretical limitations of the attack for the adaptive resonance and can be met by an attacker under reasonable assumptions.

The discussion above does not include additonal compromise elements that the attacker can carry out to guarantee the attacks success. Since the Trojan also contains a software compromise the attacker can also fix some of the parameters during the learning of the exploit. The attacker can also carry out denial of service on the benign data elements to skew the learning towards favorable compromise outcomes, among other possibilities. While these attacks can greatly improve the outcome of the adversary, it also alerts the user of the adversary's acitvities and therefore lower the stealth of the attacks. These techniques were not contemplated as part of the work, since they are not part of the learning compromise but can skew the results on the part of the adversary more than show the effects of the compromise on the learning algorithm's results.

Under the experiments carried the optimal convergence criteria were not pursued directly to obtain high degree of accuracy, since this would completely bias the experimental results for the test of hypothesis. Instead it was chosen to represent scenarios were the attacker was forced to contemplate the tradeoff of accuracy to present a broader scenario and provide random results that presents the middle bounds of accuracy versus stealth. In addition, further knowledge of the data domain was not considered in the modeled attacks which can further increase the chances of success and stealth of the adversary.

Summary

This section has presented the tests carried out on the unsupervised learning Trojan compromise. The section implemented the procedures introduced in Chapter 3 that contemplated three algorithms, eight exploits and four data sets. The data sets and exploits were divided so that each of the algorithms contained at least one data set in common with all others.

The chapter also presented individual analysis of each of tests and training runs for each of the algorithms. Several highlights were made in this chapter, especially with the relationship between the *DS* and the *DCD*. The chapter also emphasized caution on incorrectly interpreting the *DS* value, since it is a summary of the error and does not provide exact information on the specific type of error encountered during the tests.

Two hypotheses tests were carried out in this chapter. The first hypothesis test pertained to the performance of the compromises on the algorithms tested. Using the Friedman nonparametric test, it was found that there was no statistical significance in in terms of performance, among the encoding tested on the three unsupervised learning algorithms.

The second significance test involved evaluating the difference between a baseline of intercepted data samples versus an intercepted data sample with additional benign vectors generated based on a Gaussian distribution assumption. To answer this hypothesis, the Wilcoxon signed rank test was utilized. It was found that in each of the algorithms the null hypothesis had to be accepted, since the difference was not statistically significant.

The chapter also discussed the performance analysis of the compromised algorithms where a discussion of the results presented arguments to show that performance of the algorithms can in principle achieve high degree of accuracy if stealth is neglected. Under the arguments presented, it should be reasonable to assume that the attacks can be optimized by an adversary with enough accuracy can be successfully employed in real world scenarios.

Chapter 5

Conclusions, Implications, Recommendations, and Summary

Conclusions

In this research, the design of the unsupervised learning Trojan methodology was presented to highlight the importance of Trojans within the field of machine learning. This work extended the work of the Neural Network Trojan and it is part of a hypothesis that all machine learning algorithms are susceptible to Trojans.

The main problem to solve within the unsupervised learning compromise is the unsupervised learning Trojan density problem and a subproblem of the cluster identification. The unsupervised learning density problem is the general question of whether under general conditions the malicious cluster can be identified by an unsupervised learning algorithm. It was shown that this type of problem is basically a density estimation problem, and in principle, can be generally be solved.

The subproblem of cluster identification was analyzed to obtain general guidelines on the particular characteristics of the data that can be leveraged to solve the cluster identification subproblem. The challenge in finding a solution to the cluster identification problem is that the unsupervised learning algorithms are designed to separate clusters and not necessarily discriminate them. From there, an analysis was presented on intrinsic and extrinsic attributes of the payload that can be leveraged to discriminate malicious from benign data. Intrinsic properties were defined as those within a single cluster, while the extrinsic properties are those that can differentiate a malicious clusters from other benign clusters.

To show the effectiveness of the analysis, four compromises were designed to be carried out on three learning algorithms. Two of the compromises were designed for the Kohonen algorithm, another was designed for the K-means algorithm, and the final one was directed towards Adaptive Resonance. All four compromises were tested on the three algorithms under random testing, and with an emphasis on evaluating only the impact of the payload on the algorithms behavior with a less discriminative decoder, the results were moderate in success.

Under the analysis presented, it was shown that if the payload displayed extreme values such as those of the Kohonen U-Matrix encoding and big payloads under the Adaptive Resonance compromise, the attacks could be susceptible to discovery without much inspection. This presents a limitation in those types of attacks that are efficient, but provide less stealth capability for the attack. With the K-means compromise, the radius and shape of the compromise can lead to detection, but this detection would take place upon execution of a visualization technique.

The drawbacks of the particular encoding also lead to possible improvements to avoid detection. These rely on diminishing the surface area of the attack. In other words, the payloads can be selected and encoded with minimal impact on the characteristics of the data set. For example, in Adaptive Resonance, the distance from the payload can be tested to see what is the minimal distance that can lead to a minimal impact on the size of the binary encoding. With respect to K-means, the radius of the hull can be chosen to be less conspicuous.

The detection problem leads to another element of evaluation, which is the decoder of the payload. This requires access to modify the algorithm's implementation. Four modifications were presented to demonstrate the changes required to carry out the compromises documented in this research. The decoders explored do provide for particular implementations that can take into account the accuracy required to carry out the particular type of attack.

The research also presented a procedure that the adversary could use to evaluate the effectiveness of the compromises on the given data sets. The procedure consisted on analyzing an obtained sample from the expected data set to be analyzed, and use it as a training set on which to test the payload configurations. The procedure utilized a random selection of parameters to demonstrate a variety of possible scenarios which might be encountered during the execution of the procedure. Two particular aspects of the procedure were evaluated for statistical significance: the use of a particular algorithm attack on a data set and the use of a baseline versus a sample with added data points to simulate a larger dataset.

For this evaluation, the null hypothesis presented was that none of the four particular encodings on the three data sets had a particular advantage, while the alternate hypothesis sustained that the encoding did make a difference. Under the Friedman non parametric test, it was found that none of the attacks had a particular advantage over the other in obtaining better results. This does not necessarily mean that for an adversary this will be the case, as under most scenarios, the adversary will select the compromise algorithm which provides the best attack surface. It will also depend on the particular preference and ease of deployment that a particular compromise may have on a given scenario.

The second test carried out was the use of a baseline of only the intercepted sample for training, versus a training that included the intercepted data plus an additional number of samples added based on a Gaussian distribution. Each of the compromises was tested independently using the Wilcoxon ranked test and found no significant difference between them. Doing a non statistical analysis of the results of the K-means algorithm provided an example of a situation addition of samples can lead to an overestimation of the complexity of the capacities needed to drive the centroids towards the payload for activation. This effect may lead the adversary into injecting additional malicious data vectors to bias the results. In principle, this could lead to a better overall success rate, but also to a higher detection rate.

While the two tests carried out led to acceptance of the null hypothesis and that the results showed mixed success rates in the compromises, there are several elements to take into consideration when interpreting the results. The experimental setup presented in this chapter had as a goal to demonstrate a random selection of parameters to introduce the new types of attacks. The random nature of the setup is designed to show average performance behavior and does not necessarily show optimal conditions that an attacker may carry out. It was also documented that under reasonable configurations of the decoder, the attacks were substantially more successful than the bound imposed in the tests results presented. While the assumptions for the experiment followed Kerckhoffs's principle, the experimental setup itself assumed random scenarios which lead to substantial underperformance of the compromises. These factors imply that careful interpretation of the results are needed to assess the real implications of these attacks under real life scenarios.

Implications

In this work, the unsupervised learning Trojan was presented which aimed at demonstrating a proof of concept of how malware directed towards this kind of learning can be achieved. The implications of this compromise can be seen from a security perspective and from a more general perspective on machine learning.

From the security perspective, it is the first time that a malware is designed to take advantage of this type of learning. While there existed several compromises in adversarial machine learning, their aim was to subvert the learning process and not to take advantage of the algorithm to introduce a payload into a system as the ULT does.

While the research focused on showing how the compromise is achieved, its ultimate objective is to bring to light the capability that an adversary could possess, before it can actually happen in a real world scenario. This opens the possibility of research in areas such as malware detection and prevention, in order to study compromise work detection techniques that can be designed to prevent possible attacks.

The execution of the ULT under this work also gives further positive indications on the hypothesis that in principle, all machine learning algorithms are susceptible to malware compromises. The repercussions of these findings support the need for further research in other types of machine learning algorithms that may be either unsupervised, supervised, or reinforcement learning.

The work presented, along with the neural network Trojan, has further implications that go beyond the security aspects of machine learning. Learning entails not just the generalization capability, but also the capability to memorize. The insertion of the payloads which the algorithm needs to be precisely recalled, as well as having a data set which needs to be generalized, provides an example of both types of capabilities.

Recommendations and Future Work

The field of adversarial machine learning is a very important one within the general field of machine learning. As machine learning continues to gain prominence outside of the academic field and into production environments, more security concerns will arise. This need is not just based on finding out what are the possible compromises, but also on ways to detect and prevent them.

The work presented in this dissertation was delimited to a proof of concept of the unsupervised learning Trojan. This initial effort opens the door to substantial further research in the area of malware for machine learning algorithm. These areas of future research can be divided into:

1. Exploring different constraints on the access available to the attacker.
2. Refining the methodology presented.
3. Finding detection techniques for such compromises.

Each of these will be briefly discussed below.

The work presented assumed that the source code was readily available for compromise. Under certain circumstances, this may not be possible or practical for the adversary. Future work may concentrate on different methods in which the attacker could gain access to the learning algorithm, including compiled executables. Another area that merits further research is the execution of the algorithm in embedded hardware platforms and specialized circuits. Future research could include how to compromise such hardware devices without necessarily obtaining schematics or additional information on the platform.

Other constrains that merit further research are the assumptions that the attacker cannot leverage information about the data set. Research in this area may focus on how to take advantage of additional information such as sample statistics of individual fields to maximize the attack probability. It can also focus on real constraints, such as avoiding detection by adjusting the values to reasonable thresholds that the data may have within a specific data set. These constraints can also be tested under the assumption that the attacker needs minimal stealth and test the analytical assumptions proposed at the end of the previous chapter.

An assumption that was maintained throughout the procedure was the Gaussian distribution assumption on which to generate the additional benign samples. This does not necessarily reflect the distribution of the targeted data set. Future work can focus on how to analyze small samples to calculate the moments of the samples and obtain a better expanded training set on which to test the payloads.

Additional refinements to the methodology also include a more realistic assessment of the misclassified samples during the training and testing session. The addition of a loss function in future work may assign different weights to the benign data classified as malicious samples that are different to the weights assigned to malicious samples classified as benign. This will provide a better assessment of the results obtained in the confusion matrix results.

The work needs to be extended on the number of samples used for the statistical measurements to further corroborate the results presented here, along with additional unsupervised algorithms. This will support the results already obtained and expand the work on further techniques that leverage the intrinsic and extrinsic properties of the clusters. Additional work also needs to be carried on density estimation based on the obtained sample so that inclusion of the bening density measurement can provide useful measurment of the test density

The work focused on presenting the proof of concept and did not go into the critical part of detection and prevention of the attack. Two immediate mitigation techniques that can be implemented are: Improvements in code validation of machine learning software and outlier detection. Improvements to current code validation techniques require segregation of duties between the programmers who build the code for the machine learning algorithm and those that put together and run the dataset on the machine learning algorithm. Improved code walk through validation techniques taking into account possible compromises on machine learning algorithms will help mitigate any possible malicious code insertions.

Additional work on detection techniques can focus on outlier detection. The outlier detection may focus on leveraging the tradeoff between stealth and accuracy. As the attacker chooses a strategy of accuracy and neglects stealth, the attack data can be target to outlier detection. The outlier detection code can be done by an independent party and provide additional segregation of duties in the execution of the code that is used in machine learning. Further work in this area that may be considered future work is to leverage the intrinsic and extrinsic properties of utilized in the compromise to try to develop signatures for the attack. The challenge in this area is that the variability of the parameters of the intrinsic and extrinsic properties exploited by the compromises can be hard to normalize into a signature based system. Another technique that might be pursued is the use of different algorithms to pretest the data to determine whether there is a compromise in the data. This would lead to new analysis techniques on which to detect discriminating patterns, and not just identification of clusters in the data.

Summary

The Unsupervised learning Trojan is a malware insertion technique that leverages the unsupervised algorithms to carry out the activation and encoding of a malicious payload. The attack leverages either intrinsic or extrinsic properties, such as size of the sampled data, location, uniformity or density, among other possible properties to encode and activate the attack.

The unsupervised learning Trojan is a continuation on the work started on the Neural Network Trojan. Under the supervised learning of the neural network Trojan the attacker had complete control of the properties of the malware. Under the Unsupervised Learning Trojan, the adversary does not have access to a training set on which to encode the values that will be embedded on the matrix of weights as in the Neural Network Trojan. Instead, the learning Trojan poses a challenge of the Unsupervised Learning Trojan density problem. The main problem is to find a way on which the algorithm can, without any prior training, isolate the payload that is inserted as part of a benign data set into a distinct group.

Mathematically, this is equivalent to density estimation of the distributions of the targeted data set under unsupervised learning.

A sub problem within the density estimation is the malicious cluster identification. By leveraging the intrinsic and extrinsic properties of the payload, the attacker can then discriminate the malicious points within the data set's underlying distribution. The properties can also be leveraged to provide a mechanism to activate the payload upon certain threshold or embedded requirement that is integrated into the design of the unsupervised learning malware.

To analyze the Unsupervised Learning Trojan Density problem and the cluster identification subproblem, several formulas are proposed as part of generalized procedure. The procedure consists of converting an unsupervised learning execution into a "training" or calibration phase where the attacker measures the effectiveness of an attack. This training phase is based on the assumption that the adversary has intercepted part of the data that will be used by the compromised code of the algorithm, and it serves to obtain estimates of the underlying distribution of the data that will be subjected to payload insertion.

The outlined training procedure also provides a scoring example based on a compromise between execution and stealthiness. This scoring procedure can be changed by an adversary to allow for the particulars of a specific compromise. The scoring's intended goal is to choose the particular training instance that matches the attacker's main goal, whether it is malware execution guarantee or stealthiness. For example, execution guarantee may incur in additional malicious samples being injected into the data steam, therefore making the compromise more susceptible to detection.

To demonstrate the effectiveness of the procedure, four example compromises were devised on three algorithms. The algorithms chosen for the compromise were K-means, Kohonen, and Adaptive Resonance 1. Each of these algorithms had particular differences that made each of the compromises particular in nature.

For the K-means algorithm, the compromise consisted on encoding the payload as a hull, which provided the boundary of the malicious cluster. The objective was to put enough additional malicious samples within the hull, so that one of the centroids fell within the confines of the hull. The centroids' action of falling into the hull would then activate the decoder sequence to read the values of the points that comprised the hull and decode them into their final hexadecimal values.

The second and third compromises were directed at the Kohonen algorithm. The first Kohonen compromise consisted on embedding the decimal equivalent of the payload hexadecimal values into one of the set of weights of the algorithm. The sequence was then inserted into a second set of weights which provided the execution order for the payload. Finally, the compromise allowed the weights of the Kohonen algorithms to converge on these values. Once the algorithm finishes its epoch runs, the decoder is run on the particular sequence. A starting value of the sequence and payload must be given to the decoder so that the decoder can successfully read the compromise.

The second Kohonen compromise is directed at the visualization of the algorithm's output as a U-Matrix. The visualization technique compromise embeds the values of the payload into a submatrix of the Kohonen layer. When the U-Matrix visualization functions are called, the values of the Kohonen layer are processed by the U-Matrix formula which will then also decode the payload submatrix. The decoder of this compromise just needs the position of the submatrix within the Kohonen layer to read the payloads contents.

The final compromise is directed towards the Adaptive Resonance algorithm, and consisted on a similar encoding technique as the first Kohonen compromise. Two columns from the data set were identified as the targets of insertion for the payload and the sequence, respectively. The decimal values of the payload, the sequence, and the rest of the data set were then converted to binary values for processing by the Adaptive Resonance algorithm.

The four compromises were tested on a total of four data sets and eight selected payloads. For the K-means compromise, a total of 18 procedures were carried out, consisting of three training runs and one testing run. The runs used three of the four data sets and three of the selected compromises. The Kohonen compromise procedure was repeated 12 times for each compromise, using three data sets and two payloads. Finally, for Adaptive

Resonance, 12 procedures were carried out with two data sets and two payloads.

The tests carried out were configured to demonstrate a random selection of parameters to introduce the new types of attacks. This implies that some of the runs exhibited less than optimal outcomes, while other showed superior performance, which may not be indicative of possible results under real life conditions. Under real life conditions, the attacker may choose the optimal outcome based on the information that is available to him. Notwithstanding, the procedures showed a variety of outcomes that represent a wide range of scenarios, which was the main objective of the study.

Additional considerations of the outcomes of the study pertain to the chosen resolution of the decoder. The reported outcomes of the study were made with the minimum resolution of a decoder, which in most instances was suboptimal. Under the optimal configuration of the decoder, the attacks obtained substantial success rate above the reported results documented in this study. This was also part of the aims of the study, which is to provide a lower bound on the success rate of the attack.

A specific goal of the study was to test two particular hypothesis of the described procedures. The first one was to test whether the use of a particular algorithm attack on shared data set was significant. For this test, a common data set was selected among all the compromises and a Friedman significance test was implemented. It was determined under the test that there was no significant difference among the compromise chosen in terms of its performance over the others.

The second test was to determine whether there was a particular advantage of utilizing additional generated samples to simulate the final environment on which the malware will execute. The generation of additional vectors assumed a Gaussian distribution to generate the additional data elements for the simulation. It was determined, using the Wilcoxon ranked test, that there was no advantage over using only the intercepted sample by the adversary.

It should be highlighted, that while research carried out followed Kerckhoffs's principle, the experimental setup itself assumed random scenarios which lead to substantial underperformance of the compromises. The goal of the study was to establish a lower bound on performance, however the expected success rate under real life conditions are much higher, since the attacker will not choose general and random setups but ones that are optimal. This underscores the importance of future work on further experimental setups for the chosen compromises, along with other compromises targeted at other algorithms. The need for future work on detection techniques is essential to counter the threat demonstrated in this proof of concept.

While the main focus of this research was to present the Unsupervised Learning Trojan security aspect, its scope goes beyond security. Other possible implications reside in the potential to further machine learning research on problems where memorization of sequences for execution are desired, alongside the generalization properties of the algorithm. Future research may provide further insight on this and other properties of this field of study.

Appendices

A. Result Tables for K-means Experiments

Training Results Baseline Size 10683 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	111	14.23	12.80
2	1	567	27.12	43.53

Sample	Offset for Order Column	Offset for Payload Column
1	-15.44	-26.12
2	4.950	-3.86
3	69.44	-33.59

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	3199	0.24	0.0045	5.90*e-5	0.0011	11
2	0	4863	0.32	0.0046	7.89*e-5	0.0014	13
3	0	1441	0.13	0.0049	3.24*e-5	0.0017	12

Test Results for Selected Attack Using Sample 2				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	116	14.00	12.69
2	1	606	26.73	43.91

DS	*DCD*	dy_Trojan	dy_Ben
0.32	0.0013	2.26*e-5	0.0014

Confusion Matrix			
	Payload	Benign	Total
Payload	0	(124*2)+4863	(124*2)+4863
Benign	0	42730	42730

Table 10: Results of KEGG Metabolic Relation Network Data Set Baseline Using Polymorphic ShellCode - setuid(0)+setgid(0)+add user 'iph' without password to _etc_passwd

Training Results Baseline Size 10683 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	107	14.19	12.69
2	1	567	27.30	44.87

Sample	Offset for Order Column	Offset for Payload Column
1	-5.27	-64.87
2	20.80	22.01
3	28.43	53.76

Sample	Addl points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	40723	4427	0.08	0.0030	6.08*e-5	9.96*e-5	12
2	44286	24	0.99	0.0038	2.67*e-7	0.0003	11
3	34684	2654	0.94	0.0024	6.24*e-6	0.0003	13

Test Results for Selected Attack Using Sample 3				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	116	14.02	12.72
2	1	606	26.69	43.57

DS	*DCD*	dy_Trojan	dy_Ben
0	0.002	1.45*e-5	0.0017

Confusion Matrix			
	Payload	Benign	Total
Payload	(124*2)+2654	0	(124*2)+2654
Benign	0	42730	42730

Table 11: Results of KEGG Metabolic Relation Network Data Set with Additional Points using-Polymorphic ShellCode - setuid(0)+setgid(0)+add user 'iph' without password to _etc_passwd

Training Results Baseline Size 10683 of Intercepted Samples				
		Sample Statistcs		
Column	Min	Max	Mean	STD_dev
1	2	92	13.97	12.61
2	1	579	26.72	44.16

Sample	Offset for Order Column	Offset for Payload Column
1	2.66	25.21
2	-26.52	13.85
3	54.45	5.56

Sample	Addl points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	889	0.10	0.0027	0.0002	0.0001	14
2	0	3670	0.27	0.0023	3.16*e-05	0.0014	8
3	0	1500	0.14	0.0038	1.68*e-05	0.0017	14

Test Results for Selected Attack Using Sample 1(Randomly Selected)				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	116	14.07	12.75
2	1	606	26.83	43.75

DS	*DCD*	dy_Trojan	dy_Ben
0.10	0.0016	.077*e-6	0.0018

Confusion Matrix			
	Payload	Benign	Total
Payload	0	(131*2)+889	(131*2)+889
Benign	0	42730	42730

Table 12: Results of KEGG Metabolic Relation Network Data Set Baseline Using OSX_Intel reverse_tcp shell x86_64

Training Results Baseline Size 10683 of Intercepted Samples

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	2	111	14.07	12.57
2	1	606	26.098	45.00

Sample	Offset for Order Column	Offset for Payload Column
1	49.75	-1.67
2	-22.08	42.82
3	36.42	-17.65

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	47106	2460	0.96	0.0027	4.62*e-5	0.0001	10
2	28946	4003	0.10	0.0026	1.07*e-5	0.0010	8
3	31168	1966	0	0.0028	5.65*e-6	0.0009	18

Test Results for Selected Attack Using Sample 3

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	2	116	14.04	12.75
2	1	579	26.76	43.54

DS	*DCD*	dy_Trojan	dy_Ben
0	0.0018	1.23*e-5	0.0016

Confusion Matrix

	Payload	Benign	Total
Payload	(131*2)+31168	0	(131*2)+31168
Benign	0	42730	42730

Table 13: Results of KEGG Metabolic Relation Network Data Set with Additional Points using OSX_Intel reverse_tcp shell x86_64

Training Results Baseline Size 10683 of Intercepted Samples

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	2	92	13.94	12.63
2	1	509	26.32	42.67

Sample	Offset for Order Column	Offset for Payload Column
1	3.94	60.39
2	-13.83	33.83
3	12.14	16.20

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	1804	0	0.0033	0.0004	9.62*e-5	9
2	0	413	0	0.0034	7.32*e-6	0.0016	10.5
3	0	4343	0	0.0034	0.0008	8.03*e-5	10.5

Test Results for Selected Attack Using Sample 2(Randomly Chosen)

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	2	116	14.08	12.74
2	1	606	26.93	44.12

DS	*DCD*	dy_Trojan	dy_Ben
0	0.0020	1.16*e-6	0.0015

Confusion Matrix

	Payload	Benign	Total
Payload	(428*2)+413	0	(428*2)+413
Benign	0	42730	42730

Table 14: Results of KEGG Metabolic Relation Network Data Set Baseline Using Windows XP PRO SP3 - Full ROP calc shellcode

Training Results Baseline Size 10683 of Intercepted Samples

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	2	94	13.97	12.51
2	1	606	26.64	43.60

Sample	Offset for Order Column	Offset for Payload Column
1	-33.74	44.68
2	-41.71	6.42
3	-30.64	-60.05

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	27628	1466	0.95	0.0017	0.0007	0.9558	9.5
2	45021	305	0.01	0.0021	7.24*e-6	6.99*e-5	13
3	52233	3775	0.06	0.0017	4.29*e-6	0.0007	7.5

Test Results for Selected Attack Using Sample 2

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	2	94	13.98	12.51
2	1	606	26.64	43.60

DS	*DCD*	dy_Trojan	dy_Ben
0.01	0.0034	3.20*e-5	0.0001

Confusion Matrix

	Payload	Benign	Total
Payload	0	(428*2)+305	(428*2)+305
Benign	0	42730	42730

Table 15: Results of KEGG Metabolic Relation Network Data Set with Additional Points using Windows XP PRO SP3 - Full ROP calc shellcode

Training Results Baseline Size 105 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	-1	688	404.28	112.02
2	-1	60017	35847.80	9055.42

Sample	Offset for Order Column	Offset for Payload Column
1	-52.52	-171.88
2	-685.40	-17.81
3	-281.10	42.66

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	37	0.61	0.6100	1.78*e-5	2.50*e-5	8
2	0	4	0.54	0.6151	1.62*e-5	2.87*e-5	13
3	0	8	0	0.6143	1.64*e-5	2.83*e-5	15

Test Results for Selected Attack Using Sample 3				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	-1	941	401.75	130.64
2	-1	60081	35981.72	9470.95

DS	*DCD*	dy_Trojan	dy_Ben
0.24	0.6160	6.660*e-6	4.28*e-5

Confusion Matrix			
	Payload	Benign	Total
Payload	0	(124*2)+8	(124*2)+8
Benign	0	422	422

Table 16: Results of Water Treatment Plant Data Set Baseline Using Linux_x86 Polymorphic ShellCode - setuid(0)+setgid(0)+add user 'iph' without password to _etc_passwd

Training Results Baseline Size 105 of Intercepted Samples

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	-1	743	403.17	104.64
2	-1	57629	35588.37	9261.62

Sample	Offset for Order Column	Offset for Payload Column
1	-168.07	-770.24
2	158.09	243.36
3	-106.44	-393.13

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	318	26	0.26	0.5541	5.45*e-6	3.85*e-5	9
2	478	41	0.26	0.5814	6.08*e-6	4.02*e-5	12
3	372	44	0.26	0.5751	5.91*e-6	4.01*e-5	9

Test Results for Selected Attack Using Sample 2

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	-1	743	403.17	104.64
2	-1	57629	35588.37	9261.62

DS	*DCD*	dy_Trojan	dy_Ben
0	0.6151	8.11*e-6	4.04*e-5

Confusion Matrix

	Payload	Benign	Total
Payload	(124*2)+41	0	(124*2)+41
Benign	0	422	422

Table 17: Results of Water Treatment Plant Data Set with Additional Points using Linux_x86 Polymorphic ShellCode - setuid(0)+setgid(0)+add user 'iph' without password to _etc_passwd

Training Results Baseline Size of 105 Intercepted Samples				
Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	-1	815	391.28	124.68
2	-1	54578	37118.45	7616.57

Sample	Offset for Order Column	Offset for Payload Column
1	-638.04	297.62
2	93.84	269.38
3	843.95	-393.48

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	23	0	0.6814	2.08*e-5	2.28*e-5	11
2	0	26	0	0.6819	2.09*e-5	2.27*e-5	12
3	0	27	0	0.6880	2.10*e-5	2.21*e-5	13

Test Results for Selected Attack Using Sample 3				
Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	-1	941	404.98	127.61
2	-1	60081	35665.56	9759.65

DS	*DCD*	dy_Trojan	dy_Ben
0.60	0.6169	1.68*e-5	2.47*e-5

Confusion Matrix			
	Payload	Benign	Total
Payload	0	(131*2)+27	(131*2)+**27**
Benign	0	422	422

Table 18: Results of Water Treatment Plant Data Set Baseline Using OSX_Intel reverse_tcp shell x86_64

Training Results Baseline Size of 105 Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	-1	941	419.81	135.35
2	-1	52258	36235.11	9316.50

Sample	Offset for Order Column	Offset for Payload Column
1	-18.83	-67.37
2	415.64	-249.75
3	133.95	313.97

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	396	38	0.24	0.5796	5.49*e-6	4.10*e-5	9
2	391	11	0.22	0.5934	5.11*e-6	4.05*e-5	12
3	268	29	0.30	0.5533	6.38*e-6	3.69*e-5	9

Test Results for Selected Attack Using Sample 2				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	-1	887	397.88	124.65
2	-1	60081	35885.35	9406.63

DS	*DCD*	dy_Trojan	dy_Ben
0.57	0.5879	1.31*e-5	2.20*e-5

Confusion Matrix			
	Payload	Benign	Total
Payload	0	(131*2)+**11**	(131*2)+**11**
Benign	0	422	422

Table 19: Results of Water Treatment Plant Data Set with Additional Points using OSX_Intel reverse_tcp shell x86_64

Training Results Baseline Size of 105 Intercepted Samples

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	-1	815	386.96	115.871
2	-12	57606	37891.71	8073.71

Sample	Offset for Order Column	Offset for Payload Column
1	-65.73	345.85
2	264.89	464.50
3	231.59	206.75
	206.753716698854	

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	21	0	0.6574	2.972*e-5	5.48*e-6	11
2	0	16	0	0.6554	2.973*e-5	5.51*e-6	14
3	0	5	0	0.6597	2.969*e-5	5.55*e-6	11

Test Results for Selected Attack Using Sample 2

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	-1	941	406.06	129.53
2	-1	60081	35473.16	9629.18

DS	*DCD*	dy_Trojan	dy_Ben
0	0.6023	2.028*e-5	1.54*e-5

Confusion Matrix

	Payload	Benign	Total
Payload	(428*2)+16	0	(428*2)+16
Benign	0	422	422

Table 20: Results of Water Treatment Plant Data Set Baseline Using Windows XP PRO SP3 - Full ROP calc shellcode

Training Results Baseline Size of 105 Intercepted Samples

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	-1	887	404.21	143.54
2	-1	57629	36631.10	9117.38

Sample	Offset for Order Column	Offset for Payload Column
1	144.49	-135.54
2	734.99	207.75
3	702.94	48.13

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	245	4	0	0.6527	0.00043	1.16*e-5	15
2	420	32	0	0.6179	3.349*e-5	8.99*e-6	10
3	243	11	0	0.5463	3.78e-5	5.65*e-6	11

Test Results for Selected Attack Using Sample 1

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	-1	941	401.77	122.73
2	-1	60081	35786	9448.83

DS	*DCD*	dy_Trojan	dy_Ben
0	0.6156	3.73*e-5	8.96*e-6

Confusion Matrix

	Payload	Benign	Total
Payload	(428*2)+4	0	(428*2)+4
Benign	0	422	422

Table 21: Results of Water Treatment Plant Data Set with Additional Points using Windows XP PRO SP3 - Full ROP calc shellcode

Training Results Baseline Size 2900 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	71	113	85.32	8.74
2	-46	336	34.39	21.80

Sample	Offset for Order Column	Offset for Payload Column
1	83.01	3.87
2	20.67	154.31
3	76.59	105.77

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	1392	0	0.0222	0.0012	0.0002	12
2	0	228	0	0.0244	5.23*e-5	0.0004	13
3	0	836	0	0.0200	0.0001	0.0003	11

Test Results for Selected Attack Using Sample 2				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	40	118	85.37	8.88
2	-46	310	34.61	21.52

DS	*DCD*	dy_Trojan	dy_Ben
0.11	0.0016	7.11*e-06	3.31*e-05

Confusion Matrix			
	Payload	Benign	Total
Payload	0	(124*2)+228	(124*2)+228
Benign	0	14500	14500

Table 22: Results of Statlog (Shuttle) Data Set baseline using Linux_x86 Polymorphic ShellCode-setuid(0)+setgid(0)+add user 'iph' without password to _etc_passwd

Training Results Baseline Size 2900 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	72	113	85.39	9.08
2	-46	72	34.22	21.43

Sample	Offset for Order Column	Offset for Payload Column
1	-57.46	-21.50
2	24.16	201.21
3	-21.09	9.19

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	12015	1010	0.07	0.0249	9.26*e-5	7.86*e-5	9
2	11768	434	0.04	0.0252	3.52*e-5	8.25*e-5	11
3	12090	13	0.01	0.0246	1.12*e-5	8.39*e-5	10

Test Results for Selected Attack Using Sample 2				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	40	118	85.37	8.889
2	-46	310	34.61	21.53

DS	*DCD*	dy_Trojan	dy_Ben
0	0.0044	5.99*e-6	5.53*e-5

Confusion Matrix			
	Payload	Benign	Total
Payload	(124*2)+434	0	(124*2)+434
Benign	0	14500	14500

Table 23: Results of Statlog (Shuttle) Data Set with Additional Points using Linux_x86 Polymorphic ShellCode- setuid(0)+setgid(0)+add user 'iph' without password to _etc_passwd

Training Results Baseline Size 2900 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	71	113	85.51	8.89
2	-100	72	34.74	21.23

Sample	Offset for Order Column	Offset for Payload Column
1	32.76	-68.78
2	-15.98	17.86
3	-74.95	7.58

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	532	0	0.0905	0.0001	0.0013	11
2	0	1132	0	0.0935	0.0019	0.0004	12
3	0	796	0	0.1228	0.0015	0.0004	13

Test Results for Selected Attack Using Sample 2				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	40	118	85.37	8.89
2	-46	310	34.61	21.53

DS	*DCD*	dy_Trojan	dy_Ben
0	0.0044	1.99*e-5	2.58*e-5

Confusion Matrix			
	Payload	Benign	Total
Payload	(131*2)+1132	0	(131*2)+1132
Benign	0	14500	14500

Table 24: Results of Statlog (Shuttle) Data Set baseline using OSX_Intel reverse_tcp shell x86_64

Training Results Baseline Size 2900 of Intercepted Samples

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	29	141	85.23	8.90
2	-188	72	34.19	21.79

Sample	Offset for Order Column	Offset for Payload Column
1	116.50	13.32
2	-131.27	-38.37
3	-26.40	-42.30

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	8795	412	0.96	0.0309	3.54*e-5	6.96*e-5	10
2	11748	895	0.07	0.0303	4.60*e-5	6.80*e-5	11
3	8439	1373	0.12	0.0299	7.35*e-5	6.43*e-5	9

Test Results for Selected Attack Using Sample 2

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	29	141	85.23	8.90
2	-188	72	34.19	21.79

DS	*DCD*	dy_Trojan	dy_Ben
0	0.0073	1.68*e-5	2.76*e-5

Confusion Matrix

	Payload	Benign	Total
Payload	(131*2)+895	0	(131*2)+895
Benign	0	14500	14500

Table 25: Results of Statlog (Shuttle) Data Set with Additional Points using OSX_Intel reverse_tcp shell x86_64

Training Results Baseline Size 2900 of Intercepted Samples

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	71	113	85.39	8.92
2	-188	72	34.46	21.05

Sample	Offset for Order Column	Offset for Payload Column
1	79.15	0.001
2	24.14	7.37
3	-38.96	162.69

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	611	0	0.0140	3.13*e-5	0.0002	11
2	0	370	0	0.0147	7.62*e-5	8.75*e-5	15
3	0	852	0.69	0.0129	3.64*e-5	0.0002	9

Test Results for Selected Attack Using Sample 2

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	40	118	85.37	8.88
2	-46	310	34.61	21.53

DS	*DCD*	dy_Trojan	dy_Ben
0	0.0042	1.42*e-5	2.91*e-5

Confusion Matrix

	Payload	Benign	Total
Payload	(428*2)+370	0	(428*2)+370
Benign	0	14500	14500

Table 26: Results of Statlog (Shuttle) Data Set baseline using Windows XP PRO SP3 - Full ROP calc shellcode

Training Results Baseline Size 2900 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	71	149	85.37	9.06
2	-160	336	33.93	22.99

Sample	Offset for Order Column	Offset for Payload Column
1	-67.91	-96.25
2	-25.74	42.81
3	12.59	15.06

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	12065	2	0	0.0162	2.53*e-6	0.0002	12
2	10205	1028	0.90	0.0209	9.04*e-6	0.0002	13
3	11691	1173	0.91	0.0211	9.01*e-6	0.0002	11

Test Results for Selected Attack Using Sample 2				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	40	118	85.37	8.88
2	-46	310	34.61	21.52

DS	*DCD*	dy_Trojan	dy_Ben
0.67	0.0037	1.25*e-5	4.29*e-5

Confusion Matrix			
	Payload	Benign	Total
Payload	(428*2)+1028	0	(428*2)+1028
Benign	14500	0	14500

Table 27: Results of Statlog (Shuttle) Data Set with Additional Points using Windows XP PRO SP3 - Full ROP calc shellcode

Training Results Baseline Size 200 of Intercepted Samples

Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	2	65	13.48	12.21
2	1	218	23.38	34.22

Sample	Offset for Order Column	Offset for Payload Column
1	28	209
2	27	58
3	23	26

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	0	1.531	0.077	0.0002	0.0247	12.75
2	0	0	1.559	0.671	0.0002	0.0236	13.75
3	0	0	1.766	0.624	0.0002	0.0311	11.75

Test Results for Selected Attack Using Sample 2

Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	2	74	13.96	11.87
2	1	539	25.95	25.97

DS	*DCD*	dy_Trojan	dy_Ben
0.796	0.551	0.0002	0.011

Confusion Matrix			
	Payload	Benign	Total
Payload	109	3	112
Benign	26	534	560

Table 28: Results of KEGG Metabolic Relation Network Data Set Baseline Using win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode

Training Results Baseline Size 200 of Intercepted Samples

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	2	67	13.17	11.52
2	1	144	20.66	27.82

Sample	Offset for Order Column	Offset for Payload Column
1	46	143
2	25	48
3	16	2

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	505	1	0.44	0.70	0.0007	0.005	14.5
2	652	3	0.73	0.62	0.0004	0.007	13
3	654	4	1.35	0.57	0.0003	0.006	10.5

Test Results for Selected Attack Using Sample 1

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	2	74	13.98	12.04
2	1	539	26.65	45.44

DS	*DCD*	dy_Trojan	dy_Ben
0.10	0.43	0.003	0.001

Confusion Matrix

	Payload	Benign	Total
Payload	7	105	112
Benign	0	560	560

Table 29: Results of KEGG Metabolic Relation Network Data Set with Additional Points Using win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode

Training Results Baseline Size 200 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	71	14.27	12.23
2	1	270	26.09	39.95

Sample	Offset for Order Column	Offset for Payload Column
1	28	89
2	27	66
3	22	31

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	0	1.60	0.68	0.0002	0.0298	12
2	0	0	1.30	0.66	0.0003	0.0235	11.5
3	0	0	1.57	0.64	0.0003	0.0354	14.5

Test Results for Selected Attack Using Sample 3				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	84	13.20	11.88
2	1	314	24.08	37.18

DS	*DCD*	dy_Trojan	dy_Ben
1.06	0.62	0.0003	0.0226

	Confusion Matrix		
	Payload	Benign	Total
Payload	113	0	113
Benign	47	513	560

Table 30: Results of KEGG Metabolic Relation Network Data Set Baseline Using win32/xp pro sp3 (EN) 32-bit - add new local administrator

Training Results Baseline Size 200 of Intercepted Samples				
Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	2	84	13.42	12.49
2	1	217	25.97	40.83

Sample	Offset for Order Column	Offset for Payload Column
1	39	117
2	25	54
3	7	12

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	692	4	0.58	0.66	0.0004	0.0040	12.25
2	695	4	0.86	0.59	0.0003	0.0058	12.25
3	398	0	2.07	0.53	0.0002	0.0197	12.75

Test Results for Selected Attack Using Sample 3				
Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	2	71	13.41	11.82
2	1	314	24.11	36.94

DS	*DCD*	dy_Trojan	dy_Ben
0.66	0.71	0.0005	0.0048

Confusion Matrix			
	Payload	Benign	Total
Payload	111	2	113
Benign	47	513	560

Table 31: Results of KEGG Metabolic Relation Network Data Set with Additional Points Using win32/xp pro sp3 (EN) 32-bit - add new local administrator

Training Results Baseline Size 200 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	1	27.80	11.65	4.69
2	231.85	247.26	238.90	4.13

Sample	Offset for Order Column	Offset for Payload Column
1	24	243
2	16	243
3	11	233

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	0	1.12	0.65	0.0004	0.0468	16
2	0	0	1.38	0.66	0.0003	0.0411	15
3	0	0	2.03	0.64	0.0002	0.1591	7

Test Results for Selected Attack Using Sample 1				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	0.8	33.2	11.34	4.93
2	230.98	249.37	239.31	4.44

DS	*DCD*	dy_Trojan	dy_Ben
0.58	0.68	0.0009	0.0197

Confusion Matrix			
	Payload	Benign	Total
Payload	104	8	112
Benign	11	549	560

Table 32: Results of Individual Household Electric Power Consumption Data Set Baseline Using win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode

Training Results Baseline Size 200 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	1	25.40	9.39	5.04
2	231.88	246.70	240.21	3.38

Sample	Offset for Order Column	Offset for Payload Column
1	14	244
2	14	244
3	2	237

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	602	4	0.55	0.05	0.0005	0.0057	14.5
2	383	2	0.95	0.53	0.0004	0.0060	15.75
3	412	2	2.58	0.66	0.0002	0.9413	7.75

Test Results for Selected Attack Using Sample 2				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	0.8	30	9.62	5.02
2	231.61	247.08	240.21	3.18

DS	*DCD*	dy_Trojan	dy_Ben
1.00	0.69	0.0004	0.0850

Confusion Matrix			
	Payload	Benign	Total
Payload	112	0	112
Benign	42	518	560

Table 33: Results of Individual Household Electric Power Consumption Data Set with Additional Points Using win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode

Training Results Baseline Size 200 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	0.8	20	7.97	6.27
2	229.85	248.48	240.28	5.08

Sample	Offset for Order Column	Offset for Payload Column
1	18	245
2	14	245
3	4	234

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	0	1.17	0.67	0.0003	0.0620	16
2	0	0	1.80	0.69	0.0002	0.130	14.5
3	0	0	2.18	0.66	0.0002	0.121	7.5

Test Results for Selected Attack Using Sample 2				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	0.8	26	7.45	6.18
2	229.57	248.33	240.66	5.26

DS	*DCD*	dy_Trojan	dy_Ben
0.75	0.67	0.0005	0.0576

Confusion Matrix			
	Payload	Benign	Total
Payload	101	12	113
Benign	7	553	560

Table 34: Results of Individual Household Electric Power Consumption Data Set Baseline Using win32/xp pro sp3 (EN) 32-bit - add new local administrator

Training Results Baseline Size 200 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	0.8	20.6	6.74	3.74
2	235.32	247.43	241.40	3.01

Sample	Offset for Order Column	Offset for Payload Column
1	15	246
2	10	244
3	4	243

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	448	3	0.75	0.56	0.0005	0.0044	11.5
2	576	2	0.87	0.54	0.0004	0.0058	13
3	476	1	2.26	0.49	0.0002	0.0223	13.5

Test Results for Selected Attack Using Sample 3				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	0.8	21.8	6.24	3.48
2	234.19	248.32	241.63	2.978

DS	*DCD*	dy_Trojan	dy_Ben
2.41	0.65	0.0002	1.0457

Confusion Matrix			
	Payload	Benign	Total
Payload	111	2	113
Benign	52	508	560

Table 35: Results of Individual Household Electric Power Consumption Data Set with Additional Points Using win32/xp pro sp3 (EN) 32-bit - add new local administrator

Training Results Baseline Size 105 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	81	941	394.58	128.37
2	-1	55930	36041.76	9361.62

Sample	Offset for Order Column	Offset for Payload Column
1	893	45967
2	523	45404
3	198	26990

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	0	0.85	0.46	2.14*e-6	2.58*e-5	12
2	0	0	1.25	0.39	2.04*e-6	3.28*e-5	11
3	0	0	1.23	0.47	1.83*e-6	3.23*e-5	15

Test Results for Selected Attack Using Sample 3				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	-1	887	404.16	126.78
2	-1	60081	35933.46	9396.65

DS	*DCD*	dy_Trojan	dy_Ben
0.57	0.44	3.21*e-6	2.09*e-5

Confusion Matrix			
	Payload	Benign	Total
Payload	85	27	112
Benign	0	422	422

Table 36: Results of Water Treatment Data Set Baseline Using win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode

Training Results Baseline Size 105 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	-1	841	414.17	125.12
2	-1	50942	35631.84	6216.95

Sample	Offset for Order Column	Offset for Payload Column
1	630	42549
2	539	41848
3	482	14109

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	295	2	0.58	0.477	5.93*e-6	2.49*e-5	13.75
2	287	2	0.71	0.24	5.07*e-6	2.50*e-5	14.75
3	390	8	0.86	0.59	3.89*e-6	2.57*e-5	9.5

Test Results for Selected Attack Using Sample 2				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	-1	941	399.29	127.48
2	-1	60081	36035.45	10022.79

DS	*DCD*	dy_Trojan	dy_Ben
1.06	0.61	2.40*e-6	2.71*e-5

Confusion Matrix			
	Payload	Benign	Total
Payload	112	0	112
Benign	31	391	422

Table 37: Results of Water Treatment Data Set with Additional Points Using win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode

Training Results Baseline Size 105 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	81	941	408.27	124.48
2	-1	60017	36344.12	10152.93

Sample	Offset for Order Column	Offset for Payload Column
1	621	55131
2	533	46497
3	509	46413

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	0	1.03	0.84	0.0004	2.6*e-5	12
2	0	0	1.23	0.80	2.0*e-6	3.0*e-5	12
3	0	0	1.13	0.45	2.2*e-6	2.8*e-5	13

Test Results for Selected Attack Using Sample 3				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	-1	887	400.758	127.76
2	-1	60081	35858	9197.50

DS	*DCD*	dy_Trojan	dy_Ben
0.54	0.121	4.17*e-6	2.06*e-5

Confusion Matrix			
	Payload	Benign	Total
Payload	105	8	113
Benign	6	418	422

Table 38: Results of Water Treatment Data Set Baseline Using win32/xp pro sp3 (EN) 32-bit - add new local administrator

Training Results Baseline Size 105 of Intercepted Samples

Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	-1	815	389.15	133.423
2	-1	57606	35561.43	3.025

Sample	Offset for Order Column	Offset for Payload Column
1	737	50511
2	523	35564
3	287	21582

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	330	3	0.66	0.48	5.19*e-6	2.2*10-5	12.25
2	441	3	0.84	0.27	2.36*e-6	2.23*e-5	14.25
3	511	2	1.36	0.48	1.71*e-6	2.94*e-5	11.50

Test Results for Selected Attack Using Sample 2

Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	-1	941	405.51	125.33
2	-1	60081	36052.97	9036.87

DS	*DCD*	dy_Trojan	dy_Ben
0.95	0.27	2.8*e-6	2.5*e-5

Confusion Matrix			
	Payload	Benign	Total
Payload	113	0	113
Benign	45	377	422

Table 39: Results of Water Treatment Data Set with Additional Points Using win32/xp pro sp3 (EN) 32-bit - add new local administrator

Training Results Baseline Size 200 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	67	13.97	11.18
2	1	217	24.50	33.08

Sample	Offset for Order Column	Offset for Payload Column
1	18	200
2	22	209
3	17	217

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	4	0.04	2.06	0.0062	0.0016	16.5
2	0	5	0.04	2.05	0.0053	0.0015	11.5
3	0	7	0.05	2.04	0.0047	0.0016	10.5

Test Results for Selected Attack Using Sample 1				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	74	13.77	12.13
2	1	539	25.69	44.62

DS	*DCD*	dy_Trojan	dy_Ben
0.0400	1.48	0.0053	0.0006

Confusion Matrix			
	Payload	Benign	Total
Payload	5	107	112
Benign	0	560	560

Table 40: Results of KEGG Metabolic Relation Network Data Set Baseline Using win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode for additional data to carry out Friedman test

Training Results Baseline Size 200 of Intercepted Samples				
Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	2	67	14.03	13.02
2	1	539	28.12	56.00

Sample	Offset for Order Column	Offset for Payload Column
1	22	100
2	37	70
3	31	110

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	309	8	0.03	2.25	0.0079	0.0018	14.5
2	312	15	0.04	2.29	0.0080	0.0018	16.5
3	403	18	0.04	2.26	0.0077	0.0017	9

Test Results for Selected Attack Using Sample 2				
Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	2	74	13.76	11.66
2	1	499	24.79	38.46

DS	*DCD*	dy_Trojan	dy_Ben
0.055	2.72	0.0062	0.0030

Confusion Matrix			
	Payload	Benign	Total
Payload	4	108	112
Benign	0	560	560

Table 41: Results of KEGG Metabolic Relation Network Data Set with Additional Points Using win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode for additional data to carry out Friedman test

Training Results Baseline Size 200 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	57	13.11	10.69
2	1	145	21.86	28.43

Sample	Offset for Order Column	Offset for Payload Column
1	49	56
2	23	50
3	15	49

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	0	1.58	0.07	1.29*e-11	9.46*e-05	12
2	0	0	1.85	0.04	1.21*e-11	0.0008	12
3	0	0	2.13	0.11	1.01*e-11	0.0194	14

Test Results for Selected Attack Using Sample 3				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	74	13.99	12.23
2	1	539	26.35	45.38

DS	*DCD*	dy_Trojan	dy_Ben
1.66	0.089	1.36*e-11	1.82*e-7

Confusion Matrix			
	Payload	Benign	Total
Payload	132	0	132
Benign	105	455	560

Table 42: Results of KEGG Metabolic Relation Network Data Set Baseline Using win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode with U-Matrix Encoding

Training Results Baseline Size 200 of Intercepted Samples				
Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	2	65	15.83	13.03
2	1	218	28.14	37.78

Sample	Offset for Order Column	Offset for Payload Column
1	57	205
2	29	66
3	10	38

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	546	5	1.30	0.12	4.28*e-11	2.56*e-8	7.5
2	463	3	0.97	0.13	7.24*e-11	0.0005	15
3	613	2	2.17	0.20	3.57*e-11	0.0201	15.5

Test Results for Selected Attack Using Sample 3				
Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	2	74	13.31	11.60
2	1	539	24.78	43.66

DS	*DCD*	dy_Trojan	dy_Ben
2.15	0.15	3.21*e-11	0.0910

Confusion Matrix			
	Payload	Benign	Total
Payload	132	0	132
Benign	95	465	560

Table 43: Results of KEGG Metabolic Relation Network Data Set with Additional Points Using win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode with U-Matrix Encoding

Training Results Baseline Size 200 of Intercepted Samples				
Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	2	84	13.48	12.33
2	1	190	24.50	34.35

Sample	Offset for Order Column	Offset for Payload Column
1	40	174
2	26	59
3	6	14

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	0	2.28	0.07	2.88*e-11	0.0049	9
2	0	0	1.88	0.08	3.01*e-11	0.0002	15.5
3	0	0	2.56	0.08	2.31*e-11	1.11	13.5

Test Results for Selected Attack Using Sample 2				
Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	2	71	13.39	11.85
2	1	314	24.48	38.57

DS	*DCD*	dy_Trojan	dy_Ben
1.09	0.05	5.88*e-11	0.0054

Confusion Matrix			
	Payload	Benign	Total
Payload	130	2	132
Benign	67	493	560

Table 44: Results of KEGG Metabolic Relation Network Data Set Baseline Using win32/xp pro sp3 (EN) 32-bit - add new local administrator with U-Matrix Encoding

Training Results Baseline Size 200 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	71	15.01	13.61
2	1	314	31.8	51.21

Sample	Offset for Order Column	Offset for Payload Column
1	48	86
2	29	83
3	8	58

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	357	0	0.65	0.70	8.61*e-11	5.82*e-6	13
2	384	0	1	0.08	4.33*e-11	1.07*e-6	13
3	635	6	2.10	0.07	3.15*e-11	0.0006	10.5

Test Results for Selected Attack Using Sample 1(Chosen Randomly)				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	84	13.01	11.47
2	1	241	22.66	33.32

DS	*DCD*	dy_Trojan	dy_Ben
0.9	0.14	5.02*e-11	1.01*e-5

Confusion Matrix			
	Payload	Benign	Total
Payload	132	0	132
Benign	115	445	560

Table 45: Results of KEGG Metabolic Relation Network Data Set with Additional Points Using win32/xp pro sp3 (EN) 32-bit - add new local administrator with U-Matrix Encoding

Training Results Baseline Size 200 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	0.8	33.2	10.98	4.72
2	230.98	249.37	239.37	4.37

Sample	Offset for Order Column	Offset for Payload Column
1	30	245
2	16	244
3	5	237

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	0	1.66	0.08	1.25*e-11	7.03*e-5	11
2	0	0	1.89	0.66	1.17*e-11	0.0091	14
3	0	0	2.57	0.11	9.51*e-12	0.2506	13

Test Results for Selected Attack Using Sample 2				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	0.8	30.6	11.50	4.922
2	231.57	248.94	239.19	4.38

DS	*DCD*	dy_Trojan	dy_Ben
1.03	0.04	2.07*e-11	8.52*e-6

Confusion Matrix			
	Payload	Benign	Total
Payload	130	2	132
Benign	92	468	560

Table 46: Results of Individual Household Electric Power Consumption Data Set Baseline Using win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode with U-Matrix Encoding

Training Results Baseline Size 200 of Intercepted Samples

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	1	24.6	9.87	4.33
2	231.99	246.58	240.11	3.26

Sample	Offset for Order Column	Offset for Payload Column
1	21	245
2	14	243
3	11	232

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	486	2	1.28	0.11	1.68*e-11	0.0002	12.5
2	421	2	1.33	0.23	1.54*e-11	0.0003	16.75
3	510	4	1.86	0.19	1.18*e-11	8.86*e-6	8.75

Test Results for Selected Attack Using Sample 2

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	0.8	30	9.50	5.18
2	231.61	247.08	240.23	3.21

DS	*DCD*	dy_Trojan	dy_Ben
1.05	0.20	2.08*e-11	0.006

Confusion Matrix

	Payload	Benign	Total
Payload	132	0	132
Benign	35	525	560

Table 47: Results of Individual Household Electric Power Consumption Data Set with Additional Points Using win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode with U-Matrix Encoding

Training Results Baseline Size 200 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	0.8	21.2	6.38	6.21
2	229.57	248.31	241.18	5.30

Sample	Offset for Order Column	Offset for Payload Column
1	18	246
2	13	246
3	10	242

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	0	1.56	0.14	3.29*e-11	0.0004	15
2	0	0	1.96	0.11	2.94*e-11	4.09*e-6	11
3	0	0	2.01	0.01	3.08*e-11	0.0006	13

Test Results for Selected Attack Using Sample 1				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	0.8	26.6	7.73	6.18
2	230.35	248.48	240.44	5.19

DS	*DCD*	dy_Trojan	dy_Ben
0.68	0.17	6.90*e-11	3.57*e-7

Confusion Matrix			
	Payload	Benign	Total
Payload	128	4	132
Benign	37	523	560

Table 48: Results of Individual Household Electric Power Consumption Data Set Baseline Using win32/xp pro sp3 (EN) 32-bit - add new local administrator with U-Matrix Encoding

Training Results Baseline Size 200 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	0.8	16.4	6.86	3.68
2	234.53	248.32	241.58	3.05

Sample	Offset for Order Column	Offset for Payload Column
1	13	246
2	11	245
3	7	240

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	456	2	1.33	0.21	3.80*e-11	0.0006	9.5
2	482	1	1.09	0.14	4.56*e-11	0.0013	15.75
3	683	1	1.69	0.10	2.59*e-11	0.0002	12.75

Test Results for Selected Attack Using Sample 2				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	0.8	21.8	6.19	3.49
2	234.19	247.67	241.58	2.97

DS	*DCD*	dy_Trojan	dy_Ben
1.18	0.19	3.70*e-11	0.0067

Confusion Matrix			
	Payload	Benign	Total
Payload	132	0	132
Benign	70	490	560

Table 49: Results of Individual Household Electric Power Consumption Data Set with Additional Points Using win32/xp pro sp3 (EN) 32-bit - add new local administrator with U-Matrix Encoding

Training Results Baseline Size 105 of Intercepted Samples

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	81	841	410.35	127.58
2	-1	60017	37422.16	8541.91

Sample	Offset for Order Column	Offset for Payload Column
1	766	54323
2	538	45964
3	171	10514

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	0	1.62	0.93	1.07*e-12	9.86*e-8	13
2	0	0	1.82	0.52	1.13*e-12	9.47*e-8	14
3	0	0	1.54	0.05	4.45*e-12	1.60*e-6	11

Test Results for Selected Attack Using Sample 2

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	-1	941	400.24	126.96
2	-1	60081	35590.00	9544.17

DS	*DCD*	dy_Trojan	dy_Ben
0.83	0.09	2.32*e-12	3.92*e-8

Confusion Matrix

	Payload	Benign	Total
Payload	120	12	132
Benign	61	361	422

Table 50: Results of Water Treatment Data Set Baseline Using win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode with U-Matrix Encoding

Training Results Baseline Size 200 of Intercepted Samples

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	-1	841	397.89	122.15
2	-1	60081	35285.49	9820.49

Sample	Offset for Order Column	Offset for Payload Column
1	573	56447
2	520	45106
3	217	40999

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	446	5	0.87	0.62	1.80*e-12	6.51*e-8	8
2	339	3	0.91	0.04	2.12*e-12	1.84*10-5	15.5
3	451	11	0.70	0.43	2.94*e-12	2.18*e-11	14.5

Test Results for Selected Attack Using Sample 2

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	-1	941	403.34	128.34
2	-1	60017	3621.63	9272.00

DS	*DCD*	dy_Trojan	dy_Ben
0.99	0.12	1.97*e-12	2.83*e-7

Confusion Matrix

	Payload	Benign	Total
Payload	132	0	132
Benign	83	339	422

Table 51: Results of Water Treatment Data Set with Additional Points Using win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode with U-Matrix Encoding

Training Results Baseline Size 105 of Intercepted Samples				
Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	105	713	413.69	106.09
2	-1	49493	34395.04	10126.10

Sample	Offset for Order Column	Offset for Payload Column
1	705	48913
2	339	44521
3	314	30144

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	2	1.33	0.12	1.49*e-12	1.87*e-5	11.75
2	0	0	2.01	0.10	1.11*e-12	8.77*e-8	10.75
3	0	0	1.94	0.19	1.67*e-12	3.91*e-5	15.50

Test Results for Selected Attack Using Sample 3				
Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	-1	941	399.41	131.72
2	-1	60081	36343.19	9156.22

DS	*DCD*	dy_Trojan	dy_Ben
1.522	0.60	2.10*e-12	2.13*e-8

Confusion Matrix			
	Payload	Benign	Total
Payload	124	8	132
Benign	55	367	422

Table 52: Results of Water Treatment Data Set Baseline Using win32/xp pro sp3 (EN) 32-bit - add new local administrator with U-Matrix Encoding

Training Results Baseline Size 105 of Intercepted Samples				
Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	-1	713	413.63	115.12
2	-1	5528	36454.51	7835.27

Sample	Offset for Order Column	Offset for Payload Column
1	671	6957
2	529	44290
3	310	22613

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	412	7	0.62	0.04	1.69*e-11	1.06*e-8	8.25
2	340	2	1.076	0.62	1.99*e-11	3.28*e-6	16.5
3	179	7	1.85	0.07	2.33*e-12	8.85*e-7	13.25

Test Results for Selected Attack Using Sample 2				
Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	-1	941	399.42	129.82
2	-1	60081	35830.76	9734.11

DS	*DCD*	dy_Trojan	dy_Ben
0.82	0.03	2.52*e-12	4.33*e-9

Confusion Matrix			
	Payload	Benign	Total
Payload	132	0	132
Benign	61	361	422

Table 53: Results of Water Treatment Data Set with Additional Points Using win32/xp pro sp3 (EN) 32-bit - add new local administrator with U-Matrix Encoding

Training Results Baseline Size 200 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	65	11.43	12.39
2	1	218	28.86	37.88

Sample	Offset for Order Column	Offset for Payload Column
1	16	63
2	31	71
3	22	39

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	2	0.33	0.15	1.41*e-10	0.0322	8.5
2	0	0	0.26	0.17	2.30*e-10	0.0003	15
3	0	1	0.31	0.17	1.75*e-10	0.0021	15.5

Test Results for Selected Attack Using Sample 3				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	74	13.49	11.81
2	1	539	24.60	43.62

DS	*DCD*	dy_Trojan	dy_Ben
0.2655	0.16	1.98*e-10	0.0156

Confusion Matrix			
	Payload	Benign	Total
Payload	68	64	132
Benign	0	560	560

Table 54: Results of KEGG Metabolic Relation Network Data Set Baseline Using win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode with U-Matrix Encoding for additional data to carry out Friedman test

Training Results Baseline Size 200 of Intercepted Samples				
Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	2	65	14.45	12.49
2	1	147	26.45	32.83

Sample	Offset for Order Column	Offset for Payload Column
1	71	190
2	21	134
3	36	46

Sample	Addl Points	Addl Payload	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	546	9	0.33	0.16	1.44*e-10	0.0170	17
2	463	11	0.34	0.15	1.33*e-10	8.22*e-5	12.5
3	613	13	0.34	0.16	1.37*e-10	0.0185	9.5

Test Results for Selected Attack Using Sample 1				
Sample Statistics				
Column	Min	Max	Mean	STD_dev
1	2	74	13.66	11.80
2	1	539	25.20	44.67

DS	*DCD*	dy_Trojan	dy_Ben
0.3379	0.17	1.62*e-10	0.0027

Confusion Matrix			
	Payload	Benign	Total
Payload	102	30	132
Benign	0	560	560

Table 55: Results of KEGG Metabolic Relation Network Data Set with Additional Points Using win32/PerfectXp-pc1/sp3 (Tr) Add Admin Shellcode with U-Matrix Encoding for additional data to carry out Friedman test

Training Results Baseline Size 10683 of Intercepted Samples

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	2	116	13.98	12.88
2	1	509	26.40	42.15

Sample	Offset for Order Column	Offset for Payload Column
1	117	517
2	130	650
3	200	1700

Sample	Addl Points	Vigilance	*DS*	*DCD*	dy_Trojan	dy_ben_av	Score
1	0	0.065	0	1.34	0.0038	0.0019	1
2	0	0.100	0	1.63	0.0038	0.0019	2
3	0	0.200	0	1.84	0.0038	0.0019	3

Test Results for Selected Attack Using Vigilance of 0.16

Offset 117 for Order Column and 517 for Payload Column

Sample Statistics

Column	Min	Max	Mean	STD_dev
1	1	116	14.65	12.66
2	1	606	43.67	43.67

DS	*DCD*	dy_Trojan	dy_Ben
0.67	1.64	0.0012	0.00162

Confusion Matrix

	Payload	Benign	Total
Payload	18	37	51
Benign	0	42730	42730

Table 56: Results of KEGG Metabolic Relation Network Data Set Baseline Using OSX/Intel - setuid shell x86_64

Training Results Baseline Size 10683 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	111	13.89	11.58
2	1	539	25.37	44.13

Sample	Offset for Order Column	Offset for Payload Column
1	300	5000
2	275	3000
3	275	2500

Sample	Addl Points	Vigilance	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	60	0.04	0	2.17	0.0038	0.0019	2
2	100	0.100	0	2.12	0.0038	0.0019	3
3	78	0.11	0	2.29	0.0038	0.0019	1

Test Results for Selected Attack Using Vigilance of 0.19				
Offset 275 for Order Column and 2500 for Payload Column				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	116	14.01	12.65
2	1	606	26.72	43.68

DS	*DCD*	dy_Trojan	dy_Ben
0.32	1.87	0.0026	0.0016

Confusion Matrix			
	Payload	Benign	Total
Payload	37	14	51
Benign	0	42730	42730

Table 57: Results of KEGG Metabolic Relation Network Data Set with Additional Points Using OSX/Intel - setuid shell x86_64

Training Results Baseline Size 10683 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	116	14.14	12.78
2	1	567	27.23	44.36

Sample	Offset for Order Column	Offset for Payload Column
1	175	800
2	300	1200
3	300	1200

Sample	Addl Points	Vigilance	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	.044	0	2.37	0.0020	0.0017	1
2	0	.10	0	2.09	0.0020	0.0017	2
3	0	.16	0	2.41	0.0020	0.0017	3

Test Results for Selected Attack Using Vigilance of 0.022				
Offset 175 for Order Column and 800 for Payload Column				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	107	14.03	12.70
2	1	606	26.71	43.70

DS	*DCD*	dy_Trojan	dy_Ben
0	1.44	0.0020	0.0017

Confusion Matrix			
	Payload	Benign	Total
Payload	428	0	428
Benign	0	42730	42730

Table 58: Results of KEGG Metabolic Relation Network Data Set Baseline Using Windows XP PRO SP3 - Full ROP calc shellcode

Training Results Baseline Size 10683 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	111	14.07	12.81
2	1	539	26.85	43.53

Sample	Offset for Order Column	Offset for Payload Column
1	475	3000
2	375	2400
3	550	3600

Sample	Addl Points	Vigilance	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	44	0.17	0	2.64	0.002	0.0018	2
2	11	0.10	0	2.59	0.002	0.0018	1
3	147	0.18	0	1.9	0.002	0.0018	3

Test Results for Selected Attack Using Vigilance of 0.14				
Offset 375 or Order Column and 2400 for Payload Column				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	116	14.05	12.69
2	1	606	26.80	43.91

DS	*DCD*	dy_Trojan	dy_Ben
0	2.26	0.0020	0.0016

Confusion Matrix			
	Payload	Benign	Total
Payload	428	0	428
Benign	0	42730	42730

Table 59: Results of KEGG Metabolic Relation Network Data Set with Additional Points Using Windows XP PRO SP3 - Full ROP calc shellcode

Training Results Baseline Size 10683 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	111	13.97	12.69
2	1	539	26.68	43.28

Sample	Offset for Order Column	Offset for Payload Column
1	190	1300
2	257	2250
3	257	2250

Sample	Addl Points	Vigilance	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	0.042	0	1.26	0.0040	0.0018	1
2	0	0.10	0	1.62	0.0040	0.0018	2
3	0	0.088	0	1.62	0.0040	0.0018	3

Test Results for Selected Attack Using Vigilance of 0.045				
Offset 190 or Order Column and 1300 for Payload Column				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	116	14.07	43.97
2	1	606	26.84	43.97

DS	*DCD*	dy_Trojan	dy_Ben
0	1.26	0.0040	0.0016

	Confusion Matrix		
	Payload	Benign	Total
Payload	87	0	87
Benign	0	42730	42730

Table 60: Results of KEGG Metabolic Relation Network Data Set Baseline Using Windows XP SP3 English MessageBoxA Shellcode

Training Results Baseline Size 10683 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	116	14.23	12.71
2	1	579	27.13	43.94

Sample	Offset for Order Column	Offset for Payload Column
1	257	2250
2	257	2250
3	210	2050

Sample	Addl Points	Vigilance	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	186	0.14	0	2.08	0.0037	0.0017	
2	64	0.10	0	1.61	0.0037	0.0017	
3	156	0.05	0	2.25	0.0037	0.0017	

Test Results for Selected Attack Using Vigilance of 0.04				
Offset 210 or Order Column and 2050 for Payload Column				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	2	111	14.07	12.71
2	1	606	26.72	43.80

DS	*DCD*	dy_Trojan	dy_Ben
0	1.30	0.0037	0.0016

Confusion Matrix			
	Payload	Benign	Total
Payload	87	0	87
Benign	0	42730	42730

Table 61: Results of KEGG Metabolic Relation Network Data Set with Additional Points Using Windows XP SP3 English MessageBoxA Shellcode

Training Results Baseline Size 2900 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	21	113	85.32	8.99
2	-46	336	34.29	22.71

Sample	Offset for Order Column	Offset for Payload Column
1	140	550
2	140	550
3	140	550

Sample	Addl Points	Vigilance	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	.12	0	0.18	0.0039	0.0002	2
2	0	.10	0	0.18	0.0039	0.0002	1
3	0	.13	0	0.18	0.0039	0.0002	3

Test Results for Selected Attack Using Vigilance of 0.05				
Offset 140 or Order Column and 550 for Payload Column				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	40	118	85.37	8.89
2	-46	310	34.61	21.53

DS	*DCD*	dy_Trojan	dy_Ben
0	0.03	0.0039	2.38*e-05

Confusion Matrix			
	Payload	Benign	Total
Payload	51	0	51
Benign	0	14500	14500

Table 62: Statlog (Shuttle) Data Set Baseline Using OSX/Intel - setuid shell x86_64

Training Results Baseline Size 2900 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	64	113	85.14	8.81
2	-188	336	34.29	22.47

Sample	Offset for Order Column	Offset for Payload Column
1	313	1101
2	210	750
3	597	1763

Sample	Addl Points	Vigilance	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	194	.046	0	0.27	0.0038	9.33*e-5	1
2	83	0.10	0	0.16	0.0038	9.33*e-5	2
3	196	0.19	0	0.28	0.0038	9.33*e-5	3

Test Results for Selected Attack Using Vigilance of 0.19				
Offset 313 or Order Column and 1101 for Payload Column				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	40	118	85.37	8.89
2	-46	310	34.61	21.53

DS	*DCD*	dy_Trojan	dy_Ben
0.33	0.057	0.0025	2.38*e-5

Confusion Matrix			
	Payload	Benign	Total
Payload	34	17	51
Benign	0	14500	14500

Table 63: Statlog (Shuttle) Data Set with Additional Points Using OSX/Intel - setuid shell x86_64

Training Results Baseline Size 2900 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	72	113	85.40	8.82
2	-100	72	34.64	20.73

Sample	Offset for Order Column	Offset for Payload Column
1	140	550
2	120	450
3	120	245

Sample	Addl Points	Vigilance	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	0.15	0	0.18	0.0039	0.0002	3
2	0	0.10	0	0.21	0.0039	0.0002	2
3	0	0.03	0	0.07	0.0039	0.0002	1

Test Results for Selected Attack Using Vigilance of 0.16				
Offset 120 or Order Column and 245 for Payload Column				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	40	118	85.37	8.89
2	-46	310	34.61	21.53

DS	*DCD*	dy_Trojan	dy_Ben
0.78	0.015	0.0009	2.37*e-5

Confusion Matrix			
	Payload	Benign	Total
Payload	11	40	428
Benign	0	14500	14500

Table 64: Statlog (Shuttle) Data Set Baseline Using Windows XP PRO SP3 - Full ROP calc shellcode

Training Results Baseline Size 2900 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	71	113	85.33	9.00
2	-188	436	34.49	23.19

Sample	Offset for Order Column	Offset for Payload Column
1	170	2700
2	170	7100
3	170	3100

Sample	Addl Points	Vigilance	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	9	0.18	0	0.31	0.0039	6.82*e-5	3
2	73	0.10	0	0.96	0.0039	6.82*e-5	1
3	47	0.13	0	0.41	0.0039	6.82*e-5	2

Test Results for Selected Attack Using Vigilance of 0.09				
Offset 170 or Order Column and 7100 for Payload Column				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	40	118	85.37	8.89
2	-46	310	34.61	21.53

DS	*DCD*	dy_Trojan	dy_Ben
0	0.33	0.0039	2.38*e-5

	Confusion Matrix		
	Payload	Benign	Total
Payload	428	0	428
Benign	0	14500	14500

Table 65: Statlog (Shuttle) Data Set with Additional Points Using Windows XP PRO SP3 - Full ROP calc shellcode

Training Results Baseline Size 2900 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	21	113	85.22	8.62
2	-46	436	33.95	23.24

Sample	Offset for Order Column	Offset for Payload Column
1	120	630
2	120	760
3	120	840

Sample	Addl Points	Vigilance	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	0	0.07	0	0.14	0.0041	0.0001	1
2	0	0.10	0	0.17	0.0041	0.0001	2
3	0	0.17	0	0.26	0.0041	0.0001	3

Test Results for Selected Attack Using Vigilance of 0.12				
Offset 120 or Order Column and 630 for Payload Column				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	40	118	85.37	8.89
2	-46	310	34.61	21.53

DS	*DCD*	dy_Trojan	dy_Ben
0	0.03	0.0041	2.38*e-5

Confusion Matrix			
	Payload	Benign	Total
Payload	87	0	87
Benign	0	14500	14500

Table 66: Statlog (Shuttle) Data Set Baseline Using Windows XP SP3 English MessageBoxA Shellcode

Training Results Baseline Size 2900 of Intercepted Samples				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	64	141	85.55	9.02
2	-46	72	35.38	20.47

Sample	Offset for Order Column	Offset for Payload Column
1	140	520
2	140	520
3	140	12009

Sample	Addl Points	Vigilance	*DS*	*DCD*	dy_Trojan	dy_Ben	Score
1	28	0.12	0	0.10	0.0041	8.45*e-5	3
2	34	0.10	0	0.10	0.0041	8.45*e-5	2
3	37	0.04	0	1.41	0.0041	8.45*e-5	1

Test Results for Selected Attack Using Vigilance of 0.11				
Offset 140 or Order Column and 12009 for Payload Column				
		Sample Statistics		
Column	Min	Max	Mean	STD_dev
1	40	118	85.37	8.89
2	-46	310	34.61	21.53

DS	*DCD*	dy_Trojan	dy_Ben
0	0.48	0.0041	2.38*e-5

Confusion Matrix			
	Payload	Benign	Total
Payload	87	0	87
Benign	0	14500	14500

Table 67: Statlog (Shuttle) Data Set with Additional Points Using Windows XP SP3 English MessageBoxA Shellcode

References

Abidin, S., Kumar, R., & Tiwari, V. (2012). A review report on cryptovirology and cryptography. *International Journal of Scientific & Engineering Research*, *3*(11), 1.

Adleman, L. M. (1990). An abstract theory of computer viruses. *Advances in Cryptology - CRYPTO '88: Proceedings*, 1–354.

Ai, X.-W., Hu, T., Li, X., & Xiong, H. (2010). Clustering high-frequency stock data for trading volatility analysis. In *Machine Learning and Applications (ICMLA), 2010 Ninth International Conference* 333–338.

Alani, M. (2012). Neuro-cryptanalysis of DES. In *Internet Security (WorldCIS), 2012 World Congress* 23–27.

Anandrao Shivale, S. (2011). Cryptovirology: Virus approach. *International Journal of Network Security & Its Applications (IJNSA)*, *3*(4), 33–46.

Arora, A., Krishnan, R., Telang, R., & Yang, Y. (2010). An empirical analysis of software vendors' patch release behavior: Impact of vulnerability disclosure. *Information Systems Research*, *21*(1), 115–132.

Asuncion, A., & Newman, D. (n.d.). *UCI Machine Learning Repository.* Retrieved from `http://www.ics.uci.edu/~mlearn; http://epository.html`

Bacao, F., Lobo, V., & Painho, M. (2005). Self-organizing Maps as substitutes for K-Means clustering. In V. Sunderam, G. Albada, P. Sloot, & J. Dongarra (Eds.), *Computational Science – ICCS 2005* (Vol. 3516, pp. 476–483). Springer Berlin / Heidelberg. (10.1007/11428862_65)

Barreno, M., Bartlett, P. L., Chi, F. J., Joseph, A. D., Nelson, B., Rubinstein, B. I., ... Tygar, J. D. (2008). Open problems in the security of learning. In *Proceedings of the 1st ACM Workshop on AISec* 19–26.

Barreno, M., Blaine, N., Anthony, J., & Tygar, J. (2010). The security of machine learning. *Machine Learning*, *81*(2), 121–148.

Barreno, M., Nelson, B., Sears, R., Joseph, A. D., & Tygar, J. (2006). Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security* 16–25.

Battista, B., Ignazio, P., Samuel, R. B., Davide, A., Marcello, P., & Fabio, R. (in press). Is Data Clustering in Adversarial Settings Secure? In *AISec'13: Proceedings of the 2013 Artificial Intelligence and Security Workshop.*

Baumgarten, A., Steffen, M., Clausman, M., & Zambreno, J. (2011). A case study in hardware Trojan design and implementation. *International Journal of Information Security*, *10*(1), 1–14.

Bezdek, J. C., & Pal, N. R. (1998). Some new indexes of cluster validity. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions*, *28*(3), 301–315.

Bhowmik, S., & Acharyya, S. (2011). Image cryptography: The genetic algorithm approach. In *Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference* Vol. 2, 223–227.

Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., ... Roli, F. (2013). Evasion attacks against machine learning at test time. In *Machine Learning and Knowledge Discovery in Databases* (pp. 387–402). Springer.

Biggio, B., Fumera, G., Roli, F., & Didaci, L. (2012). Poisoning adaptive biometric systems. In *Structural, Syntactic, and Statistical Pattern Recognition* (pp. 417–425). Springer.

Blazejewski, A., & Coggins, R. (2004). Application of self-organizing maps to clustering of high-frequency financial data. In *Proceedings of the Second Workshop on Australasian Information Security, Data Mining and Web Intelligence, and Software Internationalisation-volume 32* 85–90.

Carpenter, G. A., & Grossberg, S. (1987a). ART 2: Self-organization of stable category recognition codes for analog input patterns. *Appl. Opt.*, *26*(23), 4919–4930.

Carpenter, G. A., & Grossberg, S. (1987b, January). A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision Graph. Image Processing* , *37*(1), 54–115.

Carpenter, G. A., & Grossberg, S. (1998). Adaptive resonance theory (ART). In M. A. Arbib (Ed.), *The handbook of brain theory and neural networks* (pp. 79–82). Cambridge, MA: MIT Press.

Chen, Q.-Z., Cheng, R., & Gu, Y.-J. (2009). Classification algorithms of Trojan horse detection based on behavior. In *Multimedia Information Networking and Security, 2009. MINES '09. International Conference* Vol. 2, pp. 510–513.

Cohen, F. (1987, February). Computer viruses: Theory and experiments. *Comput Secur.* , *6*(1), 22–35.

Cohen, F. (2001). Reply to 'Comment on" A framework for modelling Trojans and computer virus infection" ' by E. Makinen. *Computer Journal*, *44*(4), 326–327.

Cohen, P. (1995). *Empirical methods for artificial intelligence* (Vol. 139). Cambridge, Ma: MIT Press.

Dalvi, N., Domingos, P., Mausam, Sanghai, S., & Verma, D. (2004). Adversarial classification. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 99–108. New York, NY: ACM.

Davies, D. L., & Bouldin, D. W. (1979). A cluster separation measure. *Pattern Analysis and Machine Intelligence, IEEE Transactions*(2), 224–227.

Deborah, L. J., Baskaran, R., & Kannan, A. (2010). A survey on internal validity measure for cluster validation. *International Journal of Computer Science & Engineering Survey (IJCSES)*, *1*(2), 85–102.

Demsar, J. (2006). Statistical comparison of classifiers over multiple data sets. *Journal of Machine Learning*, *7*, 1–30.

Gawkowski, P., & Smulko, G. (2010). Speeding-up fault injection experiments with dynamic code injection. In *Information Technology (ICIT), 2010 2nd International Conference* 171–174.

Geigel, A. (2013). Neural network Trojan. *Journal of Computer Security*, *21*(2), 191–232.

Holz, T., Engelberth, M., & Freiling, F. (2009). Learning more about the underground economy: A case-study of keyloggers and dropzones. In *Proceedings of the 14th European Conference on Research in Computer Security* 1–18. Berlin, Heidelberg: Springer-Verlag.

Huang, L., Joseph, A. D., Nelson, B., Rubinstein, B. I., & Tygar, J. (2011). Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, 43–58.

Hughes, L. A., & DeLone, G. J. (2007). Viruses, worms, and trojan horses: Serious crimes, nuisance, or both? *Social Science Computer Review*, *25*(1), 78–98.

Ismail, I., Galal-Edeen, G., Khattab, S., & El Bahtity, M. (2012). Satellite image encryption using neural networks backpropagation. In *Computer theory and applications (ICCTA), 2012 22nd International Conference* 148–152.

Jeun, I., Lee, Y., & Won, D. (2012). A practical study on advanced persistent threats. In *Computer applications for security, control and system engineering* 144–152. Springer.

Kanter, I., Kinzel, W., & Kanter, E. (2002). Secure exchange of information by synchronization of neural networks. *Europhysics Letters*, *57*(1), 141.

Karresand, M. (2003). Separating Trojan horses, viruses, and worms - a proposed taxonomy of software weapons. In *Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society* 127–134.

Kim, M., & Ramakrishna, R. (2005). New indices for cluster validity assessment. *Pattern Recognition Letters*, *26*(15), 2353–2363.

Kloft, M., & Laskov, P. (2007). A poisoning attack against online anomaly detection. In *NIPS Workshop on Machine Learning in Adversarial Environments for Computer Security.*

Kloft, M., & Laskov, P. (2012). Security analysis of online centroid anomaly detection. *Journal of Machine Learning Research*, *13*, 3681–3724.

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

Kohonen, T. (1988). *Self-organization and associative memory.* second edition, Springer.

Kohonen, T. (1989). *Self-Organizing and associative memory* (3rd ed.). New York: Springer Verlag.

Kohonen, T. (2013). Essentials of the self-organizing map. *Neural Networks*, *37*(0), 52-65.

Kononenko, I. (2001). Machine learning for medical diagnosis: History, state of the art and perspective. *Artificial Intelligence in Medicine*, *23*, 89–109.

Krollner, B., Vanstone, B., & Finnie, G. (2010, April). Financial time series forecasting with machine learning techniques: A survey. In *Paper presented at the European Symposium on Artificial Neural Networks: Computational and Machine Learning.*

Laskov, P., & Kloft, M. (2009). A framework for quantitative security analysis of machine learning. In *Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence* 1–4. New York, NY, : ACM.

Liu, W., & Chawla, S. (2009). A game theoretical model for adversarial learning. In *Data Mining Workshops, 2009. ICDMW'09. IEEE International Conference* 25–30.

Liu, Y.-F., Zhang, L.-W., Liang, J., Qu, S., & Ni, Z.-Q. (2010). Detecting Trojan horses based on system behavior using machine learning method. In *Machine Learning and Cybernetics (ICMLC), 2010 International Conference* Vol. 2, pp. 855–860.

Lowd, D., & Meek, C. (2005). Adversarial learning. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining* 641–647. New York, NY,: ACM.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, Vol. 1, p. 14.

Mahmood, A., Dony, R., & Areibi, S. (2013). An adaptive encryption based genetic algorithms for medical images. In *Machine Learning for Signal Processing (MLSP), 2013 IEEE International Workshop* 1–6.

Mäkinen, E. (2001). Comment on 'A framework for modelling Trojans and computer virus infection'. *The Computer Journal*, *44*(4), 321–323.

McNelis, P. D. (2005). *Neural networks in finance: Gaining predictive edge in the market* Elsevier Academic Press.

Moore, T., Clayton, R., & Anderson, R. (2009). The economics of online crime. *The Journal of Economic Perspectives*, *23*(3), 3–20.

Munakata, Y., & Pfaffly, J. (2004). Hebbian learning and development. *Developmental Science*, *7*(2), 141–148.

Murphy, K. P. (2012). *Machine learning: A probabilistic perspective.* Cambridge, Ma: The MIT Press.

Nelson, B., Barreno, M., Chi, F. J., Joseph, A. D., Rubinstein, B. I., Saini, U., ... Xia, K. (2008). Exploiting machine learning to subvert your spam filter. *LEET*, *8*, 1–9.

Nelson, B., Barreno, M., Chi, F. J., Joseph, A. D., Rubinstein, B. I., Saini, U., ... Xia, K. (2009). Misleading learners: Co-opting your spam filter. In *Machine learning in cyber trust* 17–51. Springer.

Nelson, B., Biggio, B., & Laskov, P. (2011). Understanding the risk factors of learning in adversarial environments. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence* 87–92.

Nelson, B., & Joseph, A. D. (2006, June). Bounding an Attack's Complexity for a Simple Learning Model. In *Proceedings of the 1st Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SysML).*

Nelson, B., Rubinstein, B. I. P., Huang, L., Joseph, A. D., & Tygar, J. D. (2011). Classifier Evasion: Models and Open Problems. In *Proceedings of the International ECML/PKDD Conference on Privacy and Security Issues in Data Mining and Machine Learning* 92–98. Berlin: Springer-Verlag.

Nizovtsev, D., & Thursby, M. (2007). To disclose or not? An analysis of software user behavior. *Information Economics and Policy*, *19*(1), 43–64.

Plice, R. K., Pavlov, O. V., & Melville, N. P. (2008). Spam and beyond: An information-economic analysis of unwanted commercial messages. *Journal of Organizational Computing and Electronic Commerce*, *18*(4), 278–306.

Pointcheval, D. (1995). *Les Reseaux de Neurones et leurs Applications Cryptographiques* (Tech. Rep.). Technical Report, Laboratoire d'Informatique de l'Ecole Normale Superieure.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). Numerical recipes in C: The art of scientific computing 2nd edition. *Cambridge: Cambridge Univ. Press*.

Qin, J., Yan, H., Si, Q., & Yan, F. (2010). A trojan horse detection technology based on behavior analysis. In *Wireless Communications Networking and Mobile Computing (WiCOM), 2010 6th International Conference* 1–4.

Resta, M. (2009). Seize the (intra) day: Features selection and rules extraction for tradings on high-frequency data. *Neurocomputing*, *72*(16), 3413–3427.

Roli, F. B. B. F. G. (2013). Pattern recognition systems under attack. In *18th Iberoamerican Congress on Pattern Recognition (CIARP 2013).* (in press)

Sabhnani, M., & Serpen, G. (2003a). Application of machine learning algorithms to kdd intrusion detection dataset within misuse detection context. In *Proceedings of the International Conference on Machine Learning, Models, Technologies and Applications (MLMTA 2003),*

Sabhnani, M., & Serpen, G. (2003b). Application of machine learning algorithms to KDD intrusion detection dataset within misuse detection context. In *International Conference on Machine Learning, Models, Technologies and Applications 2003.*

Schneier, B. (2007). *Applied cryptography: Protocols, algorithms, and source code in C.* New York, NY, USA: John Wiley & Sons.

Simion, R. (2010). Cybercrime and its challenges between reality and fiction. Where do we actually stand? *Rivista di Criminologia, Vittimologia e Sicurezza*, *3*(1), 296–312.

Skillicorn, D. (2009). Adversarial knowledge discovery. *IEEE Intelligent Systems*, *24*, 54–61.

Stamatis, D. H. (2002). *Six sigma and beyond: Statistics and probability, Volume III* . CRC Press.

Stefanovic, P., & Kurasova, O. (2011). Visual analysis of self-organizing maps. *Nonlinear Analysis*, *16*(4), 488–504.

Stone-Gross, B., Holz, T., Stringhini, G., & Vigna, G. (2011). The underground economy of spam: A botmaster's perspective of coordinating large-scale spam campaigns. In *Proceedings of the 4th Usenix Conference on Large-scale Exploits and Emergent Threats,* 4–4. Berkeley, CA, USA: USENIX Association.

Tang, S. (2009). The detection of trojan horse based on the data mining. In *Fuzzy Systems and Knowledge Discovery, 2009. FSKD '09. Sixth International Conference* Vol. 1, 311–314.

Tehranipoor, M., & Koushanfar, F. (2010). A survey of hardware trojan taxonomy and detection. *Design Test of Computers, IEEE*, *27*(1), 10–25.

Thimbleby, H., Anderson, S., & Cairns, P. (1998). A framework for modelling trojans and computer virus infection. *The Computer Journal*, *41*(7), 444–458.

Thimbleby, H., Anderson, S., & Cairns, P. A. (2001). Reply to 'Comment on "A framework for modelling Trojans and computer virus infection"'by E. Mäkinen. *The Computer Journal*, *44*(4), 324–325.

Tsai, C.-F., Hsu, Y.-F., Lin, C.-Y., & Lin, W.-Y. (2009). Intrusion detection by machine learning: A review. *Expert Systems with Applications*, *36*(10), 11994–12000.

Verwoerd, T., & Hunt, R. (2002). Intrusion detection techniques and approaches. *Computer Communications*, *25*(15), 1356–1365.

Wackerly, D. D., III, W. M., & Scheaffer, R. L. (2002). *Mathematical statistics with applications* Sixth edition . Duxbury Advanced Series.

Webb, A. R. (2011). *Statistical pattern recognition* (3rd ed. ed.). West Sussex, England, Hoboken, NJ: John Wiley & Sons.

Wei, S., & Potkonjak, M. (2013). The undetectable and unprovable hardware Trojan horse. In *Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE* (pp. 1–2).

Whalley, I. (1999). Testing times for trojans. In *Proceedings of the Virus Bulletin Conference* Vol. 99, pp. 55–68.

Ye, H.-M., Wang, M., & Wang, T.-L. (2011). Trading rules for high-frequency financial data based on hybrid clustering. In *Fuzzy systems and knowledge discovery (fskd), 2011 Eighth International Conference* Vol. 2, pp. 1191–1195.

Young, A., & Yung, M. (1996). Cryptovirology: Extortion-based security threats and countermeasures. In *Security and privacy, 1996. Proceedings., 1996 IEEE Symposium* , 129–140.

Young, A., & Yung, M. (2004). *Malicious cryptography: Exposing cryptovirology.* Wiley. com.

Zuo, Z., & Zhou, M. (2004). Some further theoretical results about computer viruses. *The Computer Journal*, *47*(6), 627–633.

Zuo, Z., Zhu, Q.-x., & Zhou, M.-t. (2006). Infection, imitation and a hierarchy of computer viruses. *Computers & Security*, *25*(6), 469–473.