November 2021

Detecting malicious VBscripts using anomaly host based IDS based on principal component analysis (PCA)

Racha M. El-Sokkary
The American University in Cairo AUC

The American University in Cairo
School of Sciences and Engineering

Detecting Malicious VBScripts Using Anomaly Host Based IDS Based on Principal Component Analysis (PCA)

A thesis submitted to

Department of Computer Science
In partial fulfillment of the requirements for
the degree of Master of Science
in Computer Science

By
Racha Mohamed El Sokkary
B.Sc. in Computer Science

Under the supervision of
Dr. Sherif El Kassas
Prof. Dr. Amr Goneid

December 2004

The American University in Cairo

Detecting Malicious VBScripts Using Anomaly Host Based IDS Based on Principal Component Analysis (PCA)

A Thesis Submitted by Racha Mohamed El Sokkary

to Department of Computer Science

December 2004

in partial fulfillment of the requirements for
The degree of Master of Arts/Science
has been approved by

Dr. Sherif El Kassas
Thesis Committee Chair / Adviser _____
Affiliation _____
Dr. Amr Goneid
Thesis Committee Chair / Adviser _____
Affiliation _____
Dr.
Thesis Committee Reader / examiner _____
Affiliation _____
Dr.
Thesis Committee Reader / examiner _____
Affiliation _____
Dr.
Thesis Committee Reader / examiner _____
Affiliation _____

_____ _____ _____ _____
Department Chair/ Program Director Date Dean Date

DEDICATION

Dedicated to my lovely parents, my dear husband, and my daughter Jana.

ACKNOWLEDGEMENTS

The first thank you must go to my research supervisors, I would like to thank Dr. Sherif El Kassas for guiding me and helping me in all this work, without his support and active participation in every step of the process, this thesis may never have been completed. I would also like to express my sincere gratitude to Dr. Amr Goneid for sharing his experience and knowledge generously. Thank you for your valuable advice and support in my work with PCA.

I would like to acknowledge all the readers of this thesis who have given some of their precious time in reading and assessing the thesis.

Lastly, a word of thanks and appreciation must go to my husband for his support and encouragement, and to my parents for helping me in so many ways.

The American University in Cairo

ABSTRACT

Detecting Malicious VBScripts Using Anomaly Host Based IDS
Based on Principal Component Analysis (PCA)

By: Racha Mohamed El Sokkary

Thesis Supervisors: Dr. Sherif El Kassas
Prof. Dr. Amr Goneid
Department of Computer Science, AUC

Intrusion detection research over the last twenty years has focused on the threat of individuals illegally hacking into systems. Nowadays, intrusion threat to computer systems has changed radically. Instead of dealing with hackers, most current works focus on defending the system against code-driven attacks. Today's web script codes such as VBScript are receiving increasing focus as a backdoor for attacking many computers through e-mail attachments or infected web sites. The nature of these malicious codes is that they can spread widely causing serious damages to many applications. Moreover, the majority of anti-virus tools used today are able to detect known attacks but are unable to detect new and unknown attacks.

The work in this thesis presents an Anomaly host based Intrusion Detection System (IDS) that provides protection against web attacks from malicious VBScripts. The core of the system treats anomalies as outliers and this IDS model uses a Multivariate Statistical technique, Principal Component Analysis (PCA) to reduce the dimensionality of the problem while keeping the major principal components of benign instances. Hence, the system can easily filter malicious scripts that deviate from normal behavior and allow for normal scripts to bypass; so any future or unknown VBScript attacks are effectively captured while maintaining a low rate of false alarms.

Table of Contents

List of Figures

List of Tables

Chapter1: Introduction

1.1 Introduction

Computer security aims to protect an organization valuable's resources from unauthorized access, tampering of data, and denial of service. One broad definition of a secure computer system is given by Garfinkel and Spafford [7] *as "the one that can be depended upon to behave as it is expected to*". This concept can be referred to as *trust*: a system can be *trusted* if it can preserve and protect its data.

Another important definition used is that security process can be grouped into three distinct phases, such as *prevention, detection, and response* [39].

- *Prevention*: is the first line of defense for capturing known threats and preventing malicious attacks.
- *Detection*: is a continual reporting and monitoring for the appearance of new threats.
- *Response*: is the immediate action to minimize damage by locating and neutralizing the source of threats.

Thus, a secure computer system is based on the realization of *confidentiality, integrity, and availability* [4]:

- *Confidentiality* requires that information be accessible only to those authorized for it.
- *Integrity* requires that information remain unaltered by accident or malicious attempts.
- *Availability* means that the computer system remains working without denial of access and provides resources to authorized users when they need it.

Despite the widespread of computer use in the current information era with the proliferation of applications that progressively depend on e-commerce; most deployed

computer systems have become vulnerable to various kinds of attacks and break-ins. Attacks on Computer infrastructures have become a serious problem as they cause various types of damages and significant losses ranging from harming database integrity to destroying entire computer systems. According to Dorothy Denning in [1], most existing systems have security flaws that render them susceptible to intrusions, penetrations, and other forms of abuse. Threats to computer security can be categorized as in [22] to errors and omissions[1], fraud and theft[2], malicious hackers[3], and malicious codes[4].

As a society, the rapid access and processing of information over the Internet has exacerbated the problem of unauthorized access and tampering with data. Furthermore, network connectivity has increased the number and severity of these attacks. It not only provides access to larger and varied resources of data, but also provides an access path to

[1] These are important threats to data and systems' integrity. Users, data entry clerks, and systems' operators make errors that crash a system and create vulnerabilities.

[2] Individuals may use a computer to steal small amounts of money from large financial accounts. Financial systems are not the only ones at risk. Systems that control access to any material or informational resources are targets such as, inventory and school grading systems. Computer fraud and theft can be committed both by insiders (authorized users of the system) or outsiders.

[3] The term malicious hackers, sometimes called crackers, refer to those who break into computers without authorization. Much of the rise of hackers' activity is often attributed to increase in connectivity in both government and industry. The hacker threat should be considered in terms of current and potential future damage. Although current losses due to hackers' attacks are significantly smaller than losses due to insiders' theft and sabotage, the hacking problem is becoming more serious.

[4] Malicious code refers to Viruses, Worms, and Trojans that can attack both personal computers and other platforms. Viruses and worms do not need human intervention and are capable of replicating to other systems. By the time they are discovered, it may be impossible to trace their origin or the extent of the infection. Trojan horses by definition do not replicate, and are programmed to perform destructive activities triggered by a condition precompiled in its program. Actual costs attributed to the presence of malicious code results in system outages and wasting time repairing the systems.

the data from virtually anywhere on the network making the data residing on computer systems vulnerable to theft and corruption [11].

Nowadays, most intruders have become skilled at determining and exploiting systems' weaknesses to increase privileges of systems' attacks. Damaging intrusions can occur in a matter of seconds by overcoming the password authentication mechanisms designed to protect systems. Moreover, intruders can hide their presence by installing modified versions of system monitoring and administration commands and by erasing their tracks in audit logs [2], [11]. The following Figure 1. 1 describes the history of attacks and illustrates the relationship between the relative sophistication of attacks and attackers from the 1980's to the present. The knowledge required by intruders to launch known methods of attacks is decreasing. Today anyone can attack a network due to the widespread and easy availability of intrusion tools [11].

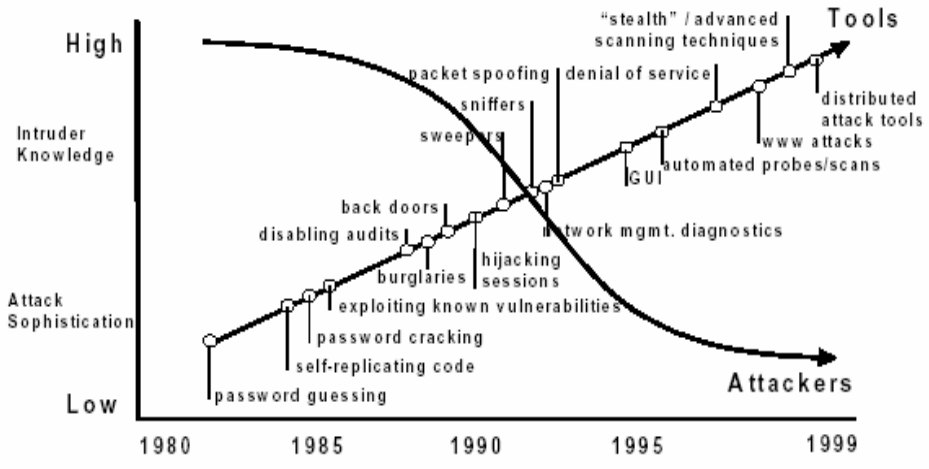


Figure 1. 1: Attack sophistication versus intruder technical knowledge [11].

Recently trends indicate that the most costly intrusions are being perpetrated by *mobile codes* rather than individual hack-in attempts. The need for dynamic content and feature-rich Web sites over the Internet has led to the adoption of mobile codes such as Java applets, Activex controls, and Script codes (e.g. Java script, VBScript, .. etc.) [23].

However, these mobile codes present a serious security threat today that merits considerable attention and resources to counteract. Security flaws and holes have been discovered in virtually every single mobile code platform, making the outcomes of their execution involve system modifications, Invasion of privacy, and denial of service [10].

One class of these mobile code threats that has proven particularly powerful and troublesome is malicious script codes. They are developed in scripting language, such as VBScript; they run on their respective interpreters residing in Web browsers or MS Office applications causing serious damages to various applications. Moreover, these attack scripts can provide hackers an easy back door entrance into the system by tricking unwitting users into downloading them from infected e-mail and web sites. In August, 2000, four of the top six malicious code threats on Symantec's Antiviral Research Center's Web page [21] were active-scripting based attacks, including: VBS.Stages.A, Wscript.KakWorm, VBS.LoveLetter, and VBS.Network [23].
These forms of malicious VBScripts are really the new types of intrusion; simply disabling their capabilities in browsers and mailers will not adequately address the problem because active scripting is used pervasively in most Web pages. That's why VBScript was chosen as an important area to investigate and focus on in this research.

Several layers of defense are used to capture malicious scripts while they are acting against the system. Firewalls[5], security patches[6], and virus scanners[7] are examples of

5 It is a system designed to prevent unauthorized access to or from a private network. Firewalls can be implemented in both hardware and software, or a combination of both. Firewalls are frequently used to prevent unauthorized Internet users from accessing private networks connected to the Internet, especially intranets. All messages entering or leaving the intranet pass through the firewall, which examines each message and blocks those that do not meet the specified security criteria [5], [6].

6 Security patches are operating systems updates to fix the security holes that malicious software exploits. Traditionally, security patches were reactively developed to address known security holes detected by attacks on the operating system. Nowadays, O/S developers employ systematic testing methodologies to discover and mitigate the system security. Similar to virus scanners, security patches protect systems only when they have been written, distributed and applied to host systems. Until then, systems remain vulnerable and attacks can spread widely [19].

these primary lines of defenses used to protect information systems from breaches and intrusions [37], [39].

In many environments, *Virus scanners* are used to protect against these malicious mobile codes. Commercial virus scanners are mostly signature based using embedded strings in software to identify malicious from benign code [21], [37]. The main drawback of virus scanners is that they are effective against known attacks but are unable to effectively detect and prevent new types of attacks. Therefore, virus scanners require frequent updating of their signature databases; otherwise the scanners become useless [37]. Today many virus scanners employ a combination of heuristics to detect viruses. The heuristics can be put into two categories: *Static heuristics and dynamic heuristics* [41]. *Static heuristics*[8] go through a file by analyzing its structure, and architecture as well as some indicators from its code to determine the likelihood of an infection. On the other hand, *dynamic heuristics*[9] set up a controlled virtual environment in which they run the program and observe its behavior to see if there is any malicious activity. The technique of maintaining a virus signature database and then checking all files against it, works pretty well for known viruses but fairs poorly against new ones simply because chances are that they would not match any of the known patterns. Thus, heuristic anti-virus software is only as good as its signature database is kept updated of new viruses [42], [43].

Computer systems are likely to remain insecure for sometime; therefore, security measures must be taken in order to detect break-ins. This motivates the need for improving detection by developing more secure systems that are resilient to attacks.
Intrusion Detection Systems (IDSs) fill this role and usually form *the last line of defense* against security breaches. They are useful not only in detecting successful break-ins, but

---

7 A type of antivirus program that searches a system for virus signatures that have attached to executable programs and applications such as e-mail clients. A virus scanner can either search all executables when a system is booted or scan a file only when a change is made to the file as viruses will change the data in a file.

8 For static heuristics, signatures are code segments of already known viruses and other patterns of malicious virus like code.

9 For dynamic heuristics, signatures are malicious requests to the operating system for patterns of malicious activity.

also in monitoring attempts to breach security. Thus, they provide important information for timely countermeasures [2].

Existing intrusion detection techniques fall in two major categories: *Misuse detection (also referred to as Signature recognition), and Anomaly detection*.

Misuse detection is similar to Virus Scanners; they also make use of *signature scanning* by looking for unique *data fingerprints* that identify certain types of attacks. The weakness of this approach is that it cannot detect novel attacks whose signatures are unknown. On the other hand, *Anomaly detection* detects both known and novel intrusions by establishing a profile of normal behavior and observing any abnormal deviation from this behavior [12].

The goal of this research is to provide protection against web script attacks from malicious VBScripts. Thus, this significant limitation of misuse detection was tackled in this research by focusing on the use of Anomaly host based IDS, so the system can easily detect VBScripts with malicious behavior onto the host and allow for normal scripts to bypass while reducing the rate of false alarms.

A key benefit of the introduced approach is that it handles a class of malicious VBScript attacks rather than specific instances, i.e. it is not a signature-based detection. As a result, it addresses future and unknown active scripting attacks. Moreover, its capability is demonstrated to detect unknown VBScript attacks in emails attachments and Web programs.

Another advantage is that the proposed anomaly host based IDS uses a Multivariate statistical technique, Principal Component Analysis (PCA) [57], to reduce the problem dimensionality by identifying patterns of benign behavior and expressing the data in such a way to highlight their similarities and differences. Thus, the IDS could easily differentiate between normal and malicious VBScripts, and capture any future malicious scripts that exhibit different or abnormal profile.

1.2 Problem Statement

It has been noticed over the past few years that the number of attacks based on malicious VBScripts has increased dramatically, making it one of the main threats to web based systems [10]. Recently there have been some high profile incidents with malicious e-mail attachments such as I LOVE YOU [40], [67], Slammer [65], and Netsky [66] viruses and their clones. These malicious VBScripts are simply transmitted as a source code in e-mail or web sites and run on the fly causing serious damages to powerful applications such as word processing, and web programs. This fact implies that VBScript codes are receiving increasing focus for attacking many computers and it is significant to provide protection against attacks from malicious VBScripts. Hence, the main problem addressed by this thesis is to develop an anomaly host based IDS that uses a multivariate statistical technique such as Principal Component Analysis (PCA) to detect attacks from malicious VBScripts.

1.3 Thesis Objective

The work presented in this thesis will address an area that received little attention despite its growing importance, namely, the increasing number of attacks utilizing security vulnerabilities in VBScript to infect many web pages [23], [45]. The objective of this research is to study the nature of these attacks as well as to propose an efficient anomaly host based system (IDS) to tackle the mentioned problem. This system is based on a classification approach utilizing Principal Component Analysis (PCA), a Multivariate Statistical Technique that identifies patterns in a large data set by highlighting their similarities and differences. The reason of choosing PCA is that the data of the mentioned problem is typically to be of high dimension, so PCA is applied to reduce the dimensionality of the data without sacrificing its valuable information. PCA is used as a feature extractor to extract patterns that will be used by the classifier, which follows the feature extractor, to differentiate between normal and malicious script. Therefore, the

introduced anomaly host based IDS could easily differentiate between normal and malicious VBScripts, and could capture any VBScript with malicious behavior.

This work is based on two important hypotheses:

* The 1st hypothesis is based on the observation that the majority of *malicious codes exhibit different behaviors in comparison to normal codes, at the same time the codes within each group share some common patterns.*

* The 2nd hypothesis assumes that *the code patterns exhibited in malicious VBScripts are unique to attack codes and record very low presence in normal VBScripts.*

The following are the steps taken to achieve the thesis objective.

1. The first step is studying the behavior of VBScript codes (Malicious and Normal script). The concise nature of VBScript language, allowed studying the normal patterns (features) of VBScript codes and detecting malicious script that deviates from normal profile to prove the validity of the 1st assumption.
2. The second step is Pattern Identification (feature extraction): this step is concerned with identifying and extracting patterns that characterize benign codes from malicious scripts. In order to do so, PCA statistical analysis is used to develop a feature extractor to extract measurements that are invariant or insensitive to detect malicious script that deviates from Normal behaviors.
3. The third step is the Attack Classifier: An attack classifier (based on Euclidean distance) is used to learn the best combination of feature elements in order to categorize the incoming script into normal or malicious code.
4. The final step is Results analysis and validation: this step accomplishes the thesis objective by analyzing the extracted results to show the validity of the hypotheses proposed in the theoretical part. So, it measures the system's efficiency such as the detection rate of malicious script and the percentage of false alarms by conducting two types of experiments for training and testing the system. Experiment over unseen VBScripts (Cross Validation) [35], and experiment over seen VBScripts.

1.4 Thesis Organization

The thesis is divided into 6 chapters. Chapter 2 describes background and puts this research into perspective. It discusses types of intrusion detection systems and requirements for efficient IDS.

Chapter 3 discusses related work in the area of anomaly host based IDS that uses multivariate statistical technique for extracting patterns for detection.

Chapter 4 describes the introduced Anomaly host based intrusion-detection system, and its components, this IDS uses Principal Component Analysis (PCA) for extracting features which are used to differentiate between Normal and Malicious VBScript.

Chapter 5 provides details of experiments and tests run on the system and the results of these tests.

Chapter 6 provides conclusions and future work expectations.

Chapter 2: Intrusion Detection

2.1 Introduction

As we increasingly rely on information infrastructures to support critical operations in banking, transportation, and telecommunication, intrusions into information systems have become a significant threat to our society with potentially severe consequences. An intrusion compromises the security of an information system through various means including denial of service, and information probing.

It is significant that the security mechanisms of any system are set up against intrusions and unauthorized access through prevention, detection, and response [39]. So, building a reliable computer system to detect intrusions resembles in many aspects a natural immune system that protects the body from dangerous foreign attackers [13]. This field of research is called Intrusion Detection; it complements prevention mechanism such as firewalls, cryptography, and authentication to capture intrusions while they are acting on the information system. This chapter will cover the concept of Intrusion Detection System and its different types. Finally, it will highlight the use of Anomaly host based IDS to prevent and detect malicious attacks.

2.2 What is an IDS

An earlier study conducted by Anderson [3] uses the term *'threat'* in the same sense as *'intrusion'* and defines it to be the potential possibility of a deliberate unauthorized attempt to:

- Access information.
- Manipulate information, or.
- Render a system unreliable or unusable.

Another definition, introduced by Edward Amoroso [8], is that *"Intrusion detection* is *the process of identifying and responding to malicious activity targeted at computing and networking resources*".

The definitions above are general enough to encompass all the threats mentioned earlier.

2.2.1 The Basic Concepts of Intrusion Detection

The concept of *'Intrusion detection'* can be clarified through the introduction of three basic, yet critical processes [8]:

Monitor: IDSs examine and process information about the target system activities. Many technical and operational issues arise in this monitoring process including timeliness of detection, confidence in information obtained, and processing power required to keep up with the monitored activity.

Report: IDSs report information about monitored systems into a system security and protection entity. This functional entity can be embedded in the monitoring component or can be conducted separately.

Respond: The purpose of intrusion detection is to reduce security risk. When related information is made available, an associated response function initiates mitigation activities. Response actions introduces myriad of factors related to the timeliness and appropriateness of the activities initiated by the IDS to deal with an incident.

2.2.2 A Generic Intrusion-Detection System

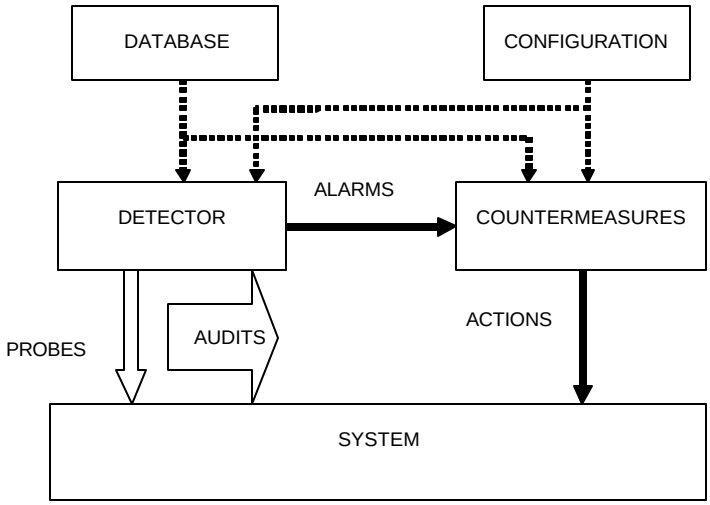


Figure 2. 1: A simple Intrusion Detection System [9].

An IDS can be described at a very macroscopic level as a detector that processes information coming into the system to be protected. As shown in Figure 2. 1, this detector uses three kinds of information: long term information related to the technique used to detect intrusions (a knowledge base of attacks), configuration information about the current state of the system, and audit information describing the events that are happening on the system [9].

The role of the detector is to eliminate unneeded information from the audit trail. It then presents a synthetic view of the current security state of the system. A decision is then taken to evaluate the probability that these states can be considered symptoms of intrusions or vulnerabilities. A countermeasure component can take corrective action to either prevent the threats from being executed or changing the state of the system back to a secure state [8], [9].

2.2.3 Characteristics of Intrusion Detection Systems

Figure 2. 2 introduces five concepts that classify IDSs [9]:

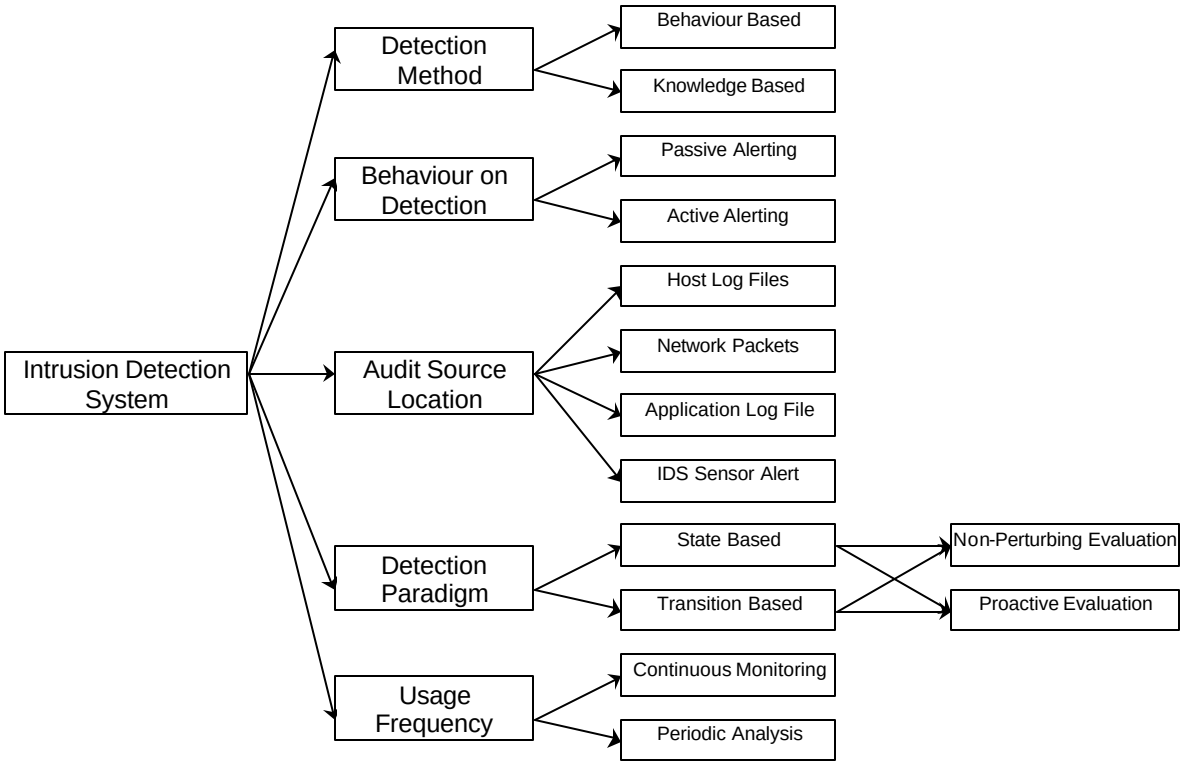


Figure 2. 2: Characteristics of Intrusion Detection Systems [9].

- ***The detection method***: describes the characteristics of the analyzer. When the IDS uses information about the normal behavior of the system, it is qualified as behavior based. Whereas when the IDS uses information about the attacks, it is qualified as knowledge based.
- ***The behavior on detection***: describes the response of the IDS to attacks. When it actively reacts to the attack by taking either corrective (closing holes), or pro-active (logging out possible attackers, closing down services) actions, then the IDS is said to be active. If the IDS merely generates alarms (including paging) it is said to be passive.

- ***The audit source location***: distinguishes IDSs based on the kind of input information they analyze. This input information can be audit trails on a host, network packets, application logs, or intrusion alerts generated by other IDSs.
- ***The detection paradigm***: describes the detection mechanism used by the IDS. IDS can evaluate states or transitions. In addition, this evaluation can be performed in a non-obtrusive way or by actively simulating the system to obtain a response.
- ***The usage frequency***: it has an orthogonal concept. Certain IDSs have real time continuous monitoring capabilities, whereas others have to be run periodically.

2.2.4 Efficiency of intrusion detection systems

Some measures are considered to evaluate the efficiency of IDSs [9]:

- ***Accuracy***: deals with the proper detection of attacks and the absence of false alarms. Inaccuracy occurs when an IDS flags as anomalous or an illegitimate action in the environment.
- ***Performance***: is the rate at which audit events are processed. If the performance of the IDS is poor, then real time detection is not possible
- ***Completeness***: is the property of an IDS to detect all attacks. Incompleteness occurs when the IDS fails to detect an attack. This measure is much more difficult to evaluate than the others because it is impossible to have a global knowledge about attacks or abuses of privileges.
- ***Fault tolerance***: IDS should be resistant to attacks, especially denial of service attacks. This is particularly important because most IDS have run above commercially available operating system or hardware, which are known to be vulnerable to attacks.
- ***Timeliness***: An IDS has to perform and propagate its analysis as quickly as possible in order to react before much damage has been done and also to prevent the attacker from subverting the audit source or the IDS itself. This implies more than the measure of performance because it, not only encompasses the processing speed of the IDS, but also the time required to propagate the information and react to it.

2.3 Classification of IDS

There are different ways to classify an IDS; intrusion detection could be classified according to *their method of detection* or according to *the type of the protected system*:

2.3.1 Classification based on the detection method [4], [9], [12]:

2.3.1.1 Misuse Detection[10] or detection by appearance.

This method is also called "knowledge-based intrusion detection" [30] because it applies the knowledge accumulated about specific attacks and system vulnerabilities and use it to identify future intrusions. Similar to virus scanners, *Misuse detection algorithms* make use of *signature scanning*, looking for unique data *fingerprints* that identify certain types of attacks. It matches activities in an information system with these signatures and signal intrusions when there is a match. This approach requires previous knowledge of an attack and is rarely effective on new attacks [34]. The accuracy of this method is considered good. However, their completeness depends on the regular update of knowledge about attacks.

- *Advantages*: This approach has very low rate of false alarms making it easy to understand the problem and take preventive or corrective measures.

- *Drawbacks*: This method has the difficulty of gathering the required information on known attacks and keeping an up to date list of all new vulnerabilities. Moreover, maintenance of the knowledge base of the intrusion detection requires careful analysis of vulnerability making it a time consuming task. Knowledge

[10] Commercial IDS systems that are widely in use today are based on these signature algorithms, such as NSM/ASIM NSM (Network Security Monitor) which is an intrusion detection system developed at the University of California-Davis, and IDES/NIDES the Intrusion Detection Expert System (IDES) developed at SRI [53], [54].

about attacks depends more on the operating system, version, platform, and application. So, the resulting IDS is closely tied to a given environment.

2.3.1.2 Anomaly Detection or detection by behavior.

This approach assumes that intrusions can be detected by observing a deviation from the normal behavior of the system or the users. The IDS compares the current activities with a model of expected behavior. When a deviation is observed, an alarm is generated. This means anything that does not correspond to a previously learned behavior, is considered intrusive. Therefore, the IDS might be complete, but judging its accuracy is a difficult issue [30].

To address this shortcoming, *Anomaly intrusion detection* look for any activity that lies outside a certain prescribed range of "safe" activities, so it captures both known and unknown intrusions if the activity under test demonstrates a significant deviation from a norm profile.

- *Advantages*: This approach can detect attempts to exploit new and unforeseen vulnerabilities. They are less dependent on the operating system. They also help to detect “abuse of privilege” types of attacks that do not actually involve exploitation of any security vulnerability.
- *Drawbacks*: The high false alarm rate is the main disadvantage of this approach because the entire scope of the behavior of an information system may not be covered during the learning phase. Moreover, the information system can undergo attacks at the same time the IDS is learning the behavior. As a result, the behavior profile will contain intrusive behavior, which is not detected as anomalous.

2.3.2 Classification based on the type of the protected system [9], [12]:

Also depending on the method that can probe the system, IDS can also be classified as *host based* or *network based* [9].

2.3.2.1 Host based systems

It bases their decision on information obtained from a single host (usually audit trails).

2.3.2.2 Network based systems

It obtains data by monitoring the traffic in the network to which the systems are connected.

2.4 Different Approaches to anomaly host-based IDS

There is extensive research work in the literature on the security of network-based anomaly IDS, such as the use of Bayesian models [28], and the use of data mining for designing network anomaly IDSs [29] as well as the security of host-based Anomaly IDS [25]. As the work in this thesis uses Anomaly host based IDS to detect malicious VBScript attacks, so the following presents previous research in the area of Anomaly host based IDS.

2.4.1 Analysis of sequences of system calls

One of the first developments in Anomaly host based intrusion detection was introduced by Stephanie Forrest's [13] which examines system processes for anomalous behavior and extracts a reference table containing all known benign sequences of system calls. These patterns are then used for live monitoring to check whether the sequences generated are listed in the table or not. If they do not, an alarm is generated. The application of this technique was shown viable for several Unix programs such as "sendmail", "lpr", and "ftpd".

A research group at the Columbia University [15] extended Forrest's work [13] by formulating machine learning tasks on operating system calls' sequences of normal and abnormal executions of the Unix sendmail program. The results show that system calls can be used to detect intrusions even with different intrusion detection algorithms.

They conducted two sets of experiments.

a. First, sequences of several consecutive system calls were extracted from sendmail traces and supplied to a machine-learning algorithm to learn the pattern of a normal sequence.

b. Second, a rule-learning program (RIPPER) was applied to the data to extract rules for predicting whether a sequence of system calls is normal or abnormal. However, the rules made by RIPPER can be erroneous. A post-processing algorithm that filters spurious prediction errors is used to scan and to determine if an intrusion has occurred.

Jones and Li [14] recently built on Forrest's method using sequences of system calls, but incorporated timing properties to create signatures that relate more closely and uniquely to the application. The time measure used is the time duration between sequence calls. Time distributions for each time interval between two calls in a unique sequence is computed, then high variance data is excluded to create a database of normal signatures that summarizes multiple measurements. The results show that by choosing appropriate analysis methods and experimentally adjusting the parameters, intrusions are effectively detected.

There are two main problems to system calls approach for host based IDS, which inhibits their use in actual deployment:

- The computational overhead for monitoring all system calls is very high which degrades the system performance.
- The system calls are irregular by nature, which makes it difficult to differentiate between normal and malicious behaviors, and increases the false positive rate.

2.4.2 Detection through the use of Neural Networks

Neural Networks (NNs)[11] have been identified as a very promising technique for addressing the high false positive rate characterizing intrusion detection systems [24]. The basic origin of this problem stems from the dynamic nature of systems and networks.

[11] Artificial neural networks (ANNs) are programs designed to simulate the way a simple biological nervous system is believed to operate. They are based on simulated nerve cells or neurons joined together in a variety of ways to form networks. These networks have the capacity to learn, memorize and create relationships amongst data [26].

It is very easy for a legitimate traffic to be mistaken for an intrusion attempt, which may yield self-inflicted denial of service incidents.

Multi layer perceptron network (MLP) [16] was proposed to examine applications at the process level in an attempt to detect anomalous behavior. The basic concept of this approach is the assumption that regardless of user characteristics and anomalous behavior at the application level, activities that are distinguishable and identifiable as anomalous will be generated at the process level as well. Results of MLP based intrusion detection systems demonstrate the suitability of using neural networks in layered detection architectures to verify alerts and minimize false positives [36].

One of the pioneer researches into the use of recurrent networks in anomaly IDS was the use of Elman networks [17], [36] in the analysis of program behavior. They are similar to the MLP with additional context nodes, which maintain a state of the system. The Elman network works by predicting the next sequence given a present input and the context. The actual next sequence is compared to the predicted sequence and the difference between them represents the measure of anomaly.

The advantages of using Neural Networks are:

- They cope well with noisy data, hence a potential reduction to false negative rate.
- Their success does not depend on any statistical assumption about the nature of the underlying data, and they are easier to modify for new user communities.

However, they have some problems.

- First, a small threshold will result in false positives while a large threshold will result in irrelevant data as well as increase the chance of false negatives.
- Second, the net topology is only determined after considerable trial and error. And third, the intruder can train the net during its learning phase

The vast majority of NNs researches into IDS have been limited to the application of the multi layer perceptron network (MLP). Only recently has the unsupervised learning model of the self-organizing map (SOM) [36], [68] been investigated but only to user behavior analysis. Therefore, future anomaly IDS researches using NN should be focused on two distinct fronts:

- The investigation of other NNs topologies, especially recurrent networks.
- Specialization of the functions of the NN detection engine.
- And the extension of NNs monitoring domains to include more system level inputs (program behavior) in addition to the present focus on user behavior analysis.

2.4.3 Detection through the use of Data Mining

There has also been a growing interest in the use of data mining techniques in Anomaly IDS to detect previously unseen malicious programs. Data mining[12] methods detect patterns such as code signatures in large amount of data and use these patterns to detect future instances in similar data [32].

A research introduced by Schultz et. al [33] aims to explore a number of standard data mining techniques over a set of training data to generate a rule based classifier to detect new malicious executables. The data mining results were compared to a traditional signature based algorithm and showed that the data mining method more than doubles the current detection rates for new malicious executables.

Another work presented in [32] uses a Malicious Email Filter (MEF) incorporated into procmail that filters multiple attachments in e-mails by using detection models obtained from the use of data mining over known malicious attachments.
The MEF system has three capabilities:

- It detects known and unknown malicious attachments.
- It provides a method for monitoring and measuring the spread of malicious attachments.
- It also allows for the efficient propagation of detection models from a central server.

[12] Data mining is a process that uses a variety of data analysis tools to discover patterns and look at numerous multidimensional data relationships concurrently, highlighting those that are dominant or exceptional [29].

One major problem in applications of data mining to intrusion detection is that detection models are produced off-line because the learning algorithms are applied to tremendous amounts of archived audit data. These models can naturally be used for *off-line intrusion detection*, or analyzing audit data off-line after intrusions have run their course. However, effective intrusion detection should happen in real-time, as intrusions take place, to minimize security compromises.

Moreover, a data mining based IDS is significantly more complex than a traditional system. The main cause for this is that data mining methods impose a heavy load on the resources of the system under protection. The use of physical memory and processing power significantly increases as the size of the database used increases, thus yielding a serious trade-off between accuracy of detection and performance of the system and its applications. Therefore, the space complexity of data mining algorithms needs to be improved without sacrificing accuracy.

2.5 Conclusion

This chapter discussed the role of Intrusion detection as an important component of the security controls and mechanisms provided in a system. IDS usually forms the last line of defense against security threats, so it detects breaches that cannot be easily detected using other methods.

It was shown that existing Intrusion detection techniques fall in two major categories: *Anomaly detection, and Misuse detection*. Misuse detection is known as a signature based system which has a limitation in that they cannot detect novel attacks. This limitation is overcome by using Anomaly detection techniques as a complement. So, Anomaly detection can detect both known and unknown attacks if they demonstrate departure from a norm profile. Two types of IDS are in use for both anomaly and misuse detection, such as *host based,* and *network based*. *Host based systems* perform host-level monitoring while *Network based systems* monitor the traffic in the network to which the systems are connected.

As the work in this thesis uses anomaly host based IDS for detecting malicious VBScript codes that deviate from the normal behavior. Thus, this chapter focuses on anomaly host based IDS as an intrusion detection technique; moreover, the next chapter will give a

survey of statistical techniques based on anomaly host based IDS that are used by other systems for detecting intrusions. It also outlines related work in the literature that studies the identification of attacks through the analysis of the source code.

Chapter 3: Related Work

3.1 Introduction

Today with the increasing complexity in business needs, most web hosts design their web pages to deliver dynamic contents and rich features to Internet clients using some of the many available Internet scripting languages. Web script codes, such as VBScript or JavaScript [10], [44], [45] are programming languages embedded in web pages to provide robust functionality to Internet applications. The increased dependence on these web scripts was met with an increased threat on the underlying systems security.

Researchers have found a number of security holes and vulnerabilities that are capable of exploiting web systems. Web script attacks, such as the propagation of malicious scripts through e-mail attachments especially new and unseen ones have rapidly increased and continue to be a leading security threat on the Internet. Recently, there have been some high profile incidents, such as "I LOVE YOU" [40], [67], Netsky [66] and its clones transmitted with malicious e-mail attachments that caused significant damage in a short time frame.

Web Script codes are transmitted as source codes over the Internet and executed on any target machine, accordingly they are hardware independent, while programs stored in binary form have limited potential to be used over the Internet due to their hardware dependence. Moreover, these script codes run mostly on the windows platform (i.e. the world's most dominant operating system) to access applications and system resources. Thus, they can independently migrate from host to host causing serious damages to various applications. For these reasons, the behavior and structure of source code attacks is receiving an increasing focus in the literature.

Based on the above, it is obvious that studying the security of web scripts such as VBScript is an important research area. The research presented in this thesis thus focuses on studying the behavior of VBScript attacks and proposes an Anomaly host-based IDS system in order to prevent potential VBScript attacks.

This chapter will review prior work in two related areas in security research that directly correlates to the proposed work in the thesis:

1- The analysis of source codes for attacks detection.
2- The use of statistical techniques (for analysis) on source code to develop an anomaly host-based IDS.

3.2 Detecting attacks through source code analysis

Source code analysis has become so significant as it is used in the assessment of code quality to find subtle defects in the code that the compiler might miss. Moreover, it is used in the analysis of Web script codes that are transmitted as source codes over the Internet and executed on the fly on different hardware platforms causing serious damages to various applications. Among the many software languages that are used for script coding (VBScript and Javascript) [23], [44] dominate the vast majority of Internet script utilization, and while they received very little attention in the literature till recently [45]. Detecting attacks of programs written in C language received significant focus [19]. The following section will highlight the main developments in the area of detecting attack source codes written in C language.

There have been a series of studies [50] that looked at modeling normal or malicious profiles by considering different properties of activity data such as:

- The frequency property (number of consecutive occurrence of individual events).
- The duration property (time of an event).
- and the ordering property (sequence or transition of events).

in order to detect intrusions. The results of these studies showed that the duration property is not sufficient to detect intrusions. Moreover, the ordering property of multiple audit events, which provides additional advantages to the frequency property for detecting intrusions are not effective unless the scalability problem of complex data is too

small. Thus, intrusion detection techniques based on the frequency property provides viable solution that produces good intrusion detection with low computational overhead. And as a result, the work presented in this thesis was chosen to focus on the frequency property in all the statistical techniques.

3.2.1 Attacks exploiting buffer-overflow systems vulnerability

Few researches have been developed to detect vulnerabilities in C language. Wagner et al. [56] have developed a system to detect buffer overflows by using static analysis in programs written in C language. Their approach treats C strings as an abstract data type accessed through the library routines and models buffers as pairs of integer ranges (size and current length), while the detection problem is formulated as an integer constraint problem. The library functions are modeled in terms of how they modify the size and length of strings. This method was concluded to be imprecise because it gives both false positives and false negatives.

One of the recent developments as well in that area was introduced by Larochelle [55] who presented a lightweight annotation assisted static analysis based on LCLint. This technique exploits information provided in programmer's semantic comments to detect buffer overflow vulnerabilities. The annotations are treated as regular C comments by the compiler, but recognized as syntactic entities by LCLint. The work was concluded to be useful in detecting that kind of attacks. A significant drawback though, is the focus on only one kind of attacks. The technique has several limitations if applied over a wider range of known attacks, such as the number of false warnings is quite high, and more annotations are needed to be inserted in the code to facilitate the checking of constraints. Moreover detection rate were not studied for testing over unknown attacks [55].

3.2.2 Attacks increasing system privileges

Cunningham and Stevenson [19] introduced a novel approach that accurately verifies script codes of UNIX attacks developed in C (or UNIX shell scripts). By detecting and differentiating those attacks that increase privileges from normal source codes. Although intrusion detection research is focused on detecting attacks after they occur, the technique

used by Cunningham and Stevenson resembles virus detection as they detect attacks before the script is run. This work has much in common with the virus detection literature. The most common technique used is signature verification, in which scripts' signatures are scanned and compared with a list of those which are known to be representative of attacks.

System Architecture

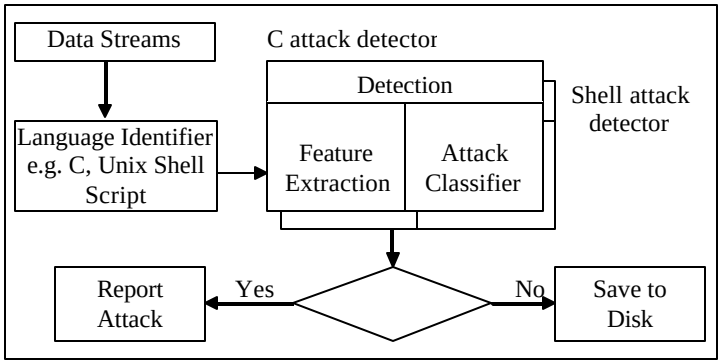


Figure 3. 1: The System Architecture of Detecting Source Code of Attacks [19].

The above Figure 3. 1 depicts Cunningham and Stevenson's proposed system architecture. The system intercepts incoming data streams and processes them to determine whether it is an attack or benign stream before reporting to the operating system. It is composed of two main functions:

1. Language identification
2. Attack detection

The language identifier is responsible for determining the language type of the incoming stream. It bases the decision on finding keywords used for the specific script language (e.g. for C, it looks for keywords such as printf or scanf). The benefit of that structure is that it allows for expanding on the system by adding identification for many languages (not just C) as proposed by the authors.

In the heart of the system lays an attack detection module which is language specific. For example, the C language detection module is triggered if the output of the language classifier shows the incoming stream to be C script. The detection module first extracts

features from the incoming script to use it for determining whether there is a potential attack or not. Cunningham and Stevenson showed that by extracting the adequate features from the script, the extracted features can be used to differentiate between malicious and benign scripts.

The feature extractor is essentially a language-specific parser that gathers statistics about the script and stores it in sets of triples (regular expression, code category, encoding scheme). Each time the regular expression records a match, the feature statistic is updated to reflect higher utilization of that specific feature.
A vector of feature statistics is then fed into a neural-network based classifier that is used to learn the best combination of feature elements in order to categorize the source into normal or malicious code. Therefore, if an attack is detected the IDS is notified, and the user can interpret the attacker's actions for the intended target.
The work in the thesis represents an extension on Cunningham's work. A system of similar architecture is used to detect attacks by monitoring VBScript source code. It uses a feature extractor for extracting features from VBScript codes and an attack classifier to determine a malicious from normal VBScript code by using statistical analysis technique such as Principal Component Analysis (PCA) [57].

3.3 Statistical-based anomaly detection techniques

Cunningham and Stevenson [19] proposed a neural-network based system in order to classify the nature of the attack. In general, once the features are extracted from the incoming data stream, there are many techniques that can be used to classify and determine whether the code is normal or malicious.
In addition to the use of neural networks, several statistical based detection techniques were studied in the literature. Such techniques use statistical properties (e.g., mean and variance) of normal activities to build a norm profile and employ statistical tests in order to determine whether the observed activities deviate significantly from the norm profile [49].
Statistical-based anomaly detection techniques are inherently capable of handling variations and noises involved in activities of information system. A norm profile

represents the features of normal activities' scripts. It should be able though to distinguish between expected variations in benign code from truly anomalous behavior.

3.3.1 Statistical process control

There are two main techniques in statistical process that are designed to handle different scenarios, but usually work together in the process control [48], [49]:

- Univariate statistical process control

  It explores each variable in a data set, separately. It looks at the range of values, as well as the central tendency of these values, so it describes the pattern of response to the variable.

  An observation is signaled if it does not conform to the normal profile, which is created using historical data. This signal may be due to a shift in the process mean and/or a shift in the process variation. Since only one variable is considered at a time, its relationship with other variables is ignored, and consequently some important information is lost.

  Earlier studies on statistical-based anomaly detection techniques are based on a statistical technique developed for (Intrusion Detection Expert System/Next Generation Intrusion Detection Expert System) IDES/NIDES [53], [54]. This is a Univariate technique that computes test statistics of a normal distribution using data on a single measure. This technique has several drawbacks [48], [49]:

  - First of all, the technique is sensitive to the normality assumption. If data on a measure are not normally distributed, the technique yields a high false alarm rate.
  - Second, it is based on a Univariate technique where a norm profile is built for only one measure of activities in information systems. On the other hand, intrusions often affect multiple measures of activities collectively. Anomalies resulting from intrusions may cause deviations on multiple measures in a collective manner rather than through separate manifestations on individual measures. Thus, Multivariate technique is a more promising area for research.

- Multivariate statistical process control

  It deals with all the variables simultaneously. Multivariate quality control methods not only can extract information on individual characteristics, but also can identify and monitor the correlation structure among them.

  A signal is generated to an event when any of the process variables falls outside the bound of process variation established by the historical data. An event is also signaled when relationships among the variables change [49].

A multivariate statistical quality monitoring is composed of two phases [48], [49]:

Phase I: Estimate process parameters.

Stage 1: *Retrospective* examination of subgroup behavior: it is the stage where historical observations are analyzed, normal behaviors are extracted and the control parameters of the process are estimated. So, a norm profile is set up with the control parameters

Stage 2: *Prospective* examination of future subgroups: it is the stage where control charts are used to detect abnormal events using the norm profile. If any anomalous phenomenon is detected during this stage, the parameters, estimated in the 1st phase, are rechecked and re-estimated to build a new norm profile. After several iterative procedures, the most suitable profile is found and used in the 2nd phase to control the process in real environment

Phase II: Use the results of the 1st phase as process parameters.

Difference between Univariate and Multivariate Technique:

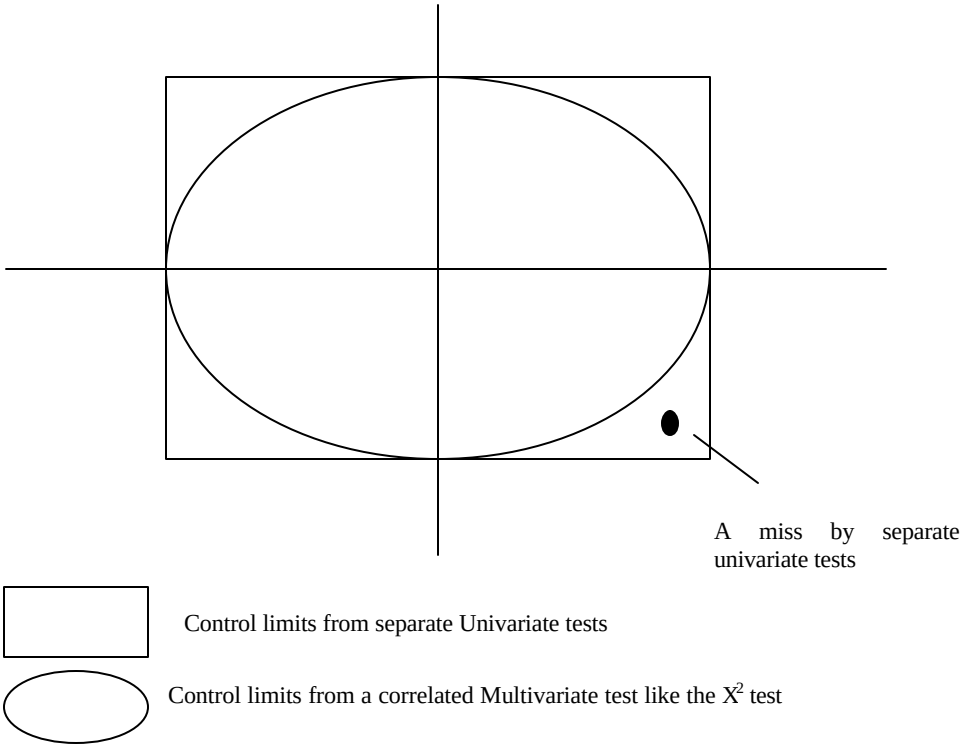


Figure 3. 2: Differences between the control limits of separate Univariate tests and the control limit of a Multivariate test [46].

Univariate control does not take into account that variables are dependent from each other and their correlation information can be very important for understanding process behavior. Thus, this method fails in detection because abnormality information is concealed inside the relationship among the variables. Moreover, it cannot detect abnormal behavior efficiently in noisy environment as well. In contrast, the multivariate control method has several benefits. It takes advantage of the correlation information and analyzes the data jointly. So, it provides an overall picture about the process change [46]. It is well understood as depicted in the above Figure 3. 2 that separate Univariate tests on individual variables can lead to misses caused by incorrect control limits of significant probabilities.

3.3.2 Multivariate statistical techniques in intrusion detection

Multivariate process control techniques that are used in Intrusion Detection systems include [46], [51]:

a. Hotelling's T^2
b. chi-squared statistics
c. and Principal Component Analysis (PCA)

All these techniques are applied to intrusion detection for monitoring and detecting anomalies of a process in an information system.

a. Hotelling's T^2 Test

Hotelling's T^2 [47] is a measure of the statistical distance from an observation to the mean estimate of the multivariate normal distribution. It is a widely used statistic for creating a multivariate control with individual observations. It combines information from all the variables in the process; so it provides an overall picture about the change of process. This technique is applied to intrusion detection for monitoring and detecting anomalies of a process in an information system.

Let us consider the computational procedure of Hotelling's T^2:

$X_i = (X_{i1}, X_{i2}, \ldots, X_{ip})$ denote an observation of P measures on a process or system at time i. When the process is operating normally (in control), the population of X follows a multivariate normal distribution with the mean vector μ and the covariance matrix S. Using a data sample of size n, the sample mean vector $\bar{x}$ in equation (1) and the sample covariance matrix S in equation (2) are usually used to estimate μ and S.

$$\bar{X} = (\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_p). \quad (1)$$

$$S = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})'. \quad (2)$$

$$T^2 = (x_i - \bar{x})s^{-1}(x_i - \bar{x}). \quad (3)$$

A large value of T^2 in equation (3) indicates a large deviation of the observation X_i from the in control population. So it signals that a statistically significant shift in the mean has occurred as soon as it is larger than a threshold limit.

Related Work

Nong ye [46] introduces the use of Hotelling's T^2 distribution to detect intrusive events by getting a norm profile when training the system with normal events. Based on this normal profile, abnormal activities can be detected in a testing event set.

Hotelling's T^2 technique detects both counter-relationship and mean shift anomalies by focusing on the individual sample mean and the sample covariance matrix, which are the main control parameters of the norm profile.

The performance of the Hotelling's T^2 test is examined on two sets of computer audit data: *a small data set and a large multiday data set*.

For the small data set, the Hotelling's T^2 test signals all the intrusion session and produces no false alarms for the normal session.

For the large data set, the Hotelling's T^2 test signals 92 percent of the intrusion sessions while producing no false alarms for the normal session.

b. The Chi square test

Chi Square test X^2 [47], [48] is another multivariate statistical analysis technique based on the chi square distribution. It is used for the 'goodness-of-fit' test because it checks whether there is a close fit between the data (O = Observed frequency) and the theory (E = Expected frequency). X^2 determines whether the differences between the observed and expected scores can be attributed to some actual difference in behavior or if this difference between the scores is a deviation from the norm profile caused by an attack.

The following illustrates the X^2 test:

Consider there are P variables to measure and X_j denotes the observation of the j^{th} variable at a particular time the X^2 test statistic is given by the equation (4).

$$x^2 = \sum_{j=1}^{p} \frac{(xj - \overline{xj})^2}{\overline{xj}} \qquad (4)$$

X^2 is the sum of squared differences between the observed and expected values of those variables. When the P variables are independent and P is large, the X^2 statistic follows approximately a normal distribution.

The X^2 distance test measures the distance of a data point from the center of a data population. Thus, it is similar to Euclidean distance [52] but uses the average value on each variable to scale the Euclidean distance on that variable or dimension.

Unlike the Hotelling's T^2 statistic which is a distance measure taking into account the covariance of multiple variables, the X^2 does not consider the relationships of multiple variables in order to simplify the computation.

Related Work

A technique based on X^2 statistic was presented by [47] that builds a norm profile of an information system and detects a large departure from the norm profile as a likely intrusion.

The results of that study show that the multivariate statistical technique based on the chi-square test statistic achieves low false alarm rate with high detection rate.

Although in this study the X^2 was tested using a small set of testing data, the results of this study demonstrate the promising potential of this technique for intrusion detection. It was concluded that this technique will be tested in the future with a large set of testing data for further evaluating of the performance and scalability.

c. Principal Components Analysis

Principal components analysis (PCA) [51], [57], [58] is used to explain the variance-covariance structure through few linear combinations of the original variables. PCA is also known as a projection method and its key objective is data reduction in a rich data environment for easy exploration and further analysis.

Work in the areas of digital signal and image processing [57], [58] found that, depending on the nature of data, it can be adequately explained using just a few factors, often far fewer than the number of original variables. Moreover, there is almost as much information in the few principal components as there is in all of the original variables

Principal components are particular linear combinations of original variable set with three important properties [47]:

1- The principal components are uncorrelated.
2- The first principal component has the highest variance; the second principal has the second highest variance and so on.
3- The total variation in all the principal components combined equal to the total variation in the original variables.

The new variables with such properties are obtained from the eigenvalues' analysis of the covariance matrix of the original variables [48].

Thus, the data overload often experienced in a rich data environment can be solved using PCA by observing the first few principal components with no significant loss of information. It is often found that PCA provides combinations of variables that are useful indicators of particular events or stages in the process.

Related Work

Cai [59] proposes the use of Principal component analysis in the detection of intrusions from malicious executables often spread as e-mail attachments. This technique outperforms many other techniques by detecting malicious executables identified as outliers that significantly deviate from the normal profile.

Shyu [51] proposes a novel scheme that uses robust principal component classifier in intrusion detection problem where the training data may be unsupervised. As anomalies can be treated as outliers, an intrusion predictive model is constructed from the major and minor principal components of normal instances. Anomalies are qualitatively different from the normal instances. That is a large deviation from the established normal patterns can be flagged as attacks. So, a measure of the difference of an anomaly from the normal instance is the distance in the principal component space.

This approach was tested on the KDD Cup 1999 [51]and it demonstrates that this method achieves high detection rate with low false alarm rate and outperforms other earlier techniques. Moreover, the statistics can be computed in less amount of time during the detection stage which makes it possible to use the method in real time.

Advantage of Principal component analysis to intrusion detection

The principal component approach to intrusion detection has some advantages [51].

First, it does not have any distributional assumption. Many statistical based intrusion detection methods assume a normal distribution which is a very general assumption and is not supported by any observation.

Second, it is typical for the data of this type of problems to be of high dimensionality. Hence, principal component analysis (PCA) is applied to reduce the dimensionality of the original data without sacrificing valuable information. Only a few parameters of principal components need to be retained for future detection as well.

3.3.3 Comparison between the different Statistical techniques

Nong ye [46] compared the results of two multivariate statistical techniques such as Hotelling's T^2, and the Chi-square X^2 test in order to analyze and detect intrusions into the host that leave trails in the audit data. As stated above, both Hotelling's T^2 test statistic and the Chi-square X^2 test measure the distance of an observation from the multivariate mean vector of a population. The Hotelling's T^2 test statistic uses the statistical distance that incorporates both mean shifts and counter-relationships (the relationships of multiple variables) whereas the Chi-square X^2 test detects only mean shifts.

The results of this study show that the performance of Hotelling's T^2 test for intrusion detection is not compensated by its intensive computation in comparison to the performance of the Chi-square X^2 test that detects only mean shifts. The Chi-square X^2 performed well in intrusion detection, it signaled all the intrusion session and produced no false alarms on the normal sessions. Therefore, it appears that a Chi-square X^2 multivariate analysis technique detecting mean shifts is only sufficient and effective for intrusion detection than The Hotelling's T^2 test. This is due to the nature of intrusion detection problem which distinguishes between normal events from intrusive events by capturing any departure from the normal profile i.e. any deviation of an observation from the mean as an anomaly or likely intrusion. Thus, intrusion is manifested more through focusing on detecting mean shifts rather than the relationships of multiple variables.

Moreover, the computational complexity of the Chi-square X^2 test is much less than the Hotelling's T^2 test.

3.4 Conclusion

This chapter focused on two important areas related to the work presented in the thesis. First, it reviews previous work in the area of detecting attacks through source codes analysis. Second, it describes statistical based anomaly detection technique such as Univariate and Multivariate statistical technique.

It points how Multivariate technique provides an overall picture about the process change than Univariate technique which does not take into account that variables are dependent of each other. Thus, it fails in detection because abnormality information is concealed inside the relationship among the variables.

Multivariate statistical techniques can be classified into distance-based techniques, and variance-covariance relation based techniques.

Distance-based techniques use the distance between the mean normal profile and the current observation value in order to detect intrusions. There are many distance metrics that fall under this category such as the Euclidean distance, Chi-square, and Canberra metric [52].

Variance-covariance relation based techniques use a measure that takes into account the covariance of multiple variables (the relationship of multi-variables). Hotelling's T^2 and Principal Component Analysis fall under this category. However, Principal Component Analysis provides a greater advantage in comparison to Hotelling's T^2 because it reduces the data dimensionality without sacrificing information content. So, instead of dealing with hundreds of variables, a few principal components are used.

Based on the above, the work on the thesis will focus on using Principal Component Analysis (PCA) as a statistical technique to reduce data dimensionality of the variables (features) in VBScript codes.

Chapter 4: Development of an Anomaly Host Based IDS

4.1 Introduction

Rapid development in Internet technology provided people with a convenient way to exchange large amounts of information. However, the Internet is basically an insecure environment mainly because its native design lacks the presence of security barriers to prevent malicious attacks. For that reason, the need for Internet security systems grew as fast as the dependence on the Internet to conduct business transactions.

Another area that received significant attention as well was the detection of attacks through the analysis of the source code of the incoming data streams. This technique is most useful in identifying attacks embedded in Shell and C source codes (in UNIX environment).

The use of script languages, especially VBScripts [23], [44] on web pages is now a common practice. VBScript attacks continue to be a leading security threat on the Internet. A VBScript virus, such as "Loveletter" [40] caused global havoc by embedding its code in HTML email to hide its malicious potential. Moreover, VBScript is a very easy script language to program. Accordingly, writing a VBScript virus nowadays is becoming an extremely easy task in comparison to the old use of assemblers requiring specialized and highly skilled programmers.

As a result of the above discussion, the work presented in this thesis addresses an area that received little attention despite its growing importance. This chapter will present an anomaly host based IDS that efficiently identifies VBScript attacks. As stated earlier in chapter 3, this IDS model represents an extension on a similar research line followed by Cunningham [19], which detects script codes of UNIX attacks developed in C or shell script code that increase privileges. The introduced technique uses a classifier based on a

Multivariate statistical technique, Principal Component Analysis (PCA) [57], to identify patterns of benign behavior and to express the data in such a way to highlight their similarities and differences. So, the IDS could easily differentiate between normal and malicious VBScripts, and could capture any future malicious scripts that exhibit different or abnormal patterns from a profile of normal behavior.

The analysis followed to develop the IDS was divided into the following phases:

1. Malicious code study: this phase studies the behavior of malicious VBScript codes and how malicious patterns deviate from normal behavior. The outcome of this study was useful in identifying the key differences between patterns of benign and malicious scripts.
2. Pattern Identification (feature extractor): in that stage, the concern was to identify patterns that characterize benign codes. Hence, a large data set was collected to extract features that identify the common themes across the code. This problem is tackled by using an efficient multivariate statistical technique such as PCA statistical analysis as a feature extractor.
3. Attack Classifier: an implementation for an attack classifier was proposed. This classifier is used to learn the best combination of feature elements in order to categorize the incoming script into normal or malicious code.
4. Results analysis and validation: this phase analyzes the results to show the validity of the hypotheses proposed.

4.2 Malicious code study

Script mobile codes are usually written in scripting language, such as VBScript or Java script. VBScript language was developed by Microsoft as a competitor to Java script for automating web pages; the most obvious difference between them is only in their syntax [10]. My preference to study and focus on VBScript is due to the fact that VBScript is receiving increasing focus as a serious threat for attacking many computers [10], [44]. To illustrate the size of the problem associated with the lack of security in mobile VBScript codes, it is remarkable to realize that in 1999; only 2 malicious VBScript were publicly

known [10]. Nowadays there are more than 4000 registered malicious VBScript and the toll increases everyday [20], [21].

This fact implies that VBScript codes are receiving increasing focus as a backdoor for attacking computers by threatening the user's computer through sending itself through e-mail attachment or through browsing an infected site. The core of the malicious script, which contains the destructive part of the virus, can restrict its activity to unnoticeable level by changing the windows registry, causing data's loss, and spreading itself to infect other systems.

4.2.1 Data Collection

In order to study the behavior of malicious attacks, it was necessary to collect information around the nature of these attacks, the different methods of infections, and propagations. This necessitated getting hand on the source code of VBScript attacks in order to understand the specific behaviors exhibited by attack codes.

The following represents various sources used in the process of gathering information around VBScript attacks,

1. Antivirus sites (e.g. McAfee, Symantec) [20], [21].
   Antivirus sites contain a wealth of information around the different viral attacks. They provide a comprehensive analysis on the behavior of the different attacks and its execution flow.
2. Private security sites (e.g. Ebcvg.com Your source for information security)[60].
   This was the main source used in gathering attack source codes. Over the internet, there are some sites specialized in serving information security. Many of these sites focus on building an up-to-date list of source codes of current attacks and they offer these sources for sale.
3. Virus collectors' sites (e.g. VX Heavens)[61].
   Amateur sites were used to top up the gathered virus collection through exchanging Virii source codes with other collectors.

Due to security measures over the Internet, getting a large sample of malicious VBScripts represents a difficult task because many hackers' sites or virus collectors' are rapidly closed. Moreover, displaying security vulnerabilities or holes on the Web could be used

as a conduit by other malicious hackers or rogue programs to steal data or wreak other damages. So, the total number of collected malicious VBScripts from private sites and virus collectors to represent the malicious VBScript code population in this work are in the range of 283 malicious VBScripts (see Appendix D).

The group of malicious VBScripts under study covers many of the known attack techniques,

1. Scripts that infect Vbs files and reside in the directories c:\, c:\windows, c:\windows\desktop, c:\my documents, and c:\startup.
2. Sometimes these malignant scripts insert infected Vbs/doc/htm, or Vba macro virus.
3. or propagate via MS-Outlook.

On the other hand, in order to study the normal behavior of VBScript codes, approximately 1113 general-purpose VBScripts (see Appendix D) were collected representing around 3 times the attack code collection. This collection was gathered to cover a wide range of applications such as [44],

1. Maintaining Disk and file systems, files and folders, logs, printing, processes, registry, and services.
2. Providing security such as verifying script signature.
3. Task scheduling.
4. Creating users accounts and groups
5. And other multi purpose script collected from Microsoft technet script center.

4.2.2 Attack pattern analysis

The foundation of all attack detection techniques to date is based on the ability to differentiate between normal and malicious behaviors. This implicitly suggests that *all malicious codes exhibit different behaviors in comparison to benign codes, at the same time codes within each group share common patterns* (initially referred to as the *1st hypothesis* stated earlier in Chapter1).

The first part of the above assumption was verified by analyzing the collected Malicious VBScript codes, as it was apparent that malicious script codes can take distinctive forms that differentiate them from normal codes.

a. Types of attack codes

According to the method of attack, malicious VBScript codes can be classified into 3 main categories: namely viruses, worms, and Trojan horses [20], [21], [23].

1. *A Virus* is a program that causes an unexpected event by infecting and modifying other programs. It is also capable of self-execution and replication.
2. *A Worm* is a program that resides in the active memory of a computer and duplicates itself. It may send copies to other computers through some communication applications (e.g. Internet Relay Chat - IRC).
3. *A Trojan Horse* is a malicious program that pretends to be a benign application, but does unexpected destruction to different programs. Trojans are unlike viruses since they do not replicate.

It is noted from the above classification that these categories of malicious codes are not mutually exclusive. For instance, a malicious program could be both a worm and a virus or a worm could be delivered as a Trojan. Hence, these forms of attacks cover all known malicious VBScripts.

b. Life cycle of attack code

In order to verify *the 2nd part of the assumption* (*i.e., 'attack codes share common patterns')*, it is necessary to understand more about the life cycle of attack codes. The CERIAS Intrusion Detection Research Group [62] analyzed the behavior of several worms, and developed a general life-cycle that describes the different stages that an attack code passes through. The life cycle from the point of view of the victim host consists of four stages as shown in Figure 4. 1:

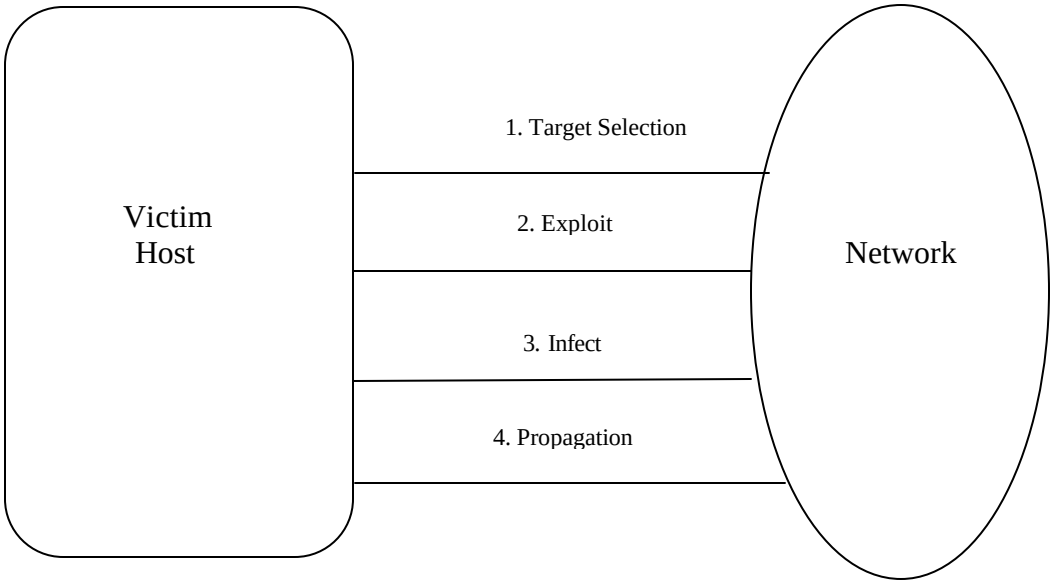


Figure 4. 1: The Life Cycle of Attack Codes.

1. Target selection

The target selection stage is the phase when an uninfected host is chosen for attack. This is where the malicious code or worm performs reconnaissance to determine other potential victims.

2. Exploitation

The exploitation phase is when the attacker compromises the target by exploiting a particular vulnerability (or convincing the victim to execute the code). Worms use well known vulnerabilities and published exploits to compromise their target.

3. Infection

The infection stage is significant in this life-cycle, as the malicious code copies itself on to the victim machine and then performs any number of different actions. The line between the infection and exploitation stage is

sometimes blurred. For example, the Code Red worm[13] [62]actually loads itself into the victim's memory as part of the exploit used to compromise the victim. The infection stage is understood to be the time when the worm "sets up shop" on the newly infected machine. For example, the worm can open backdoors, change system files, or attempt to hide its presence by replacing system utilities with Trojan horses.

4. Propagation

In the propagation stage, the malicious code attempts to spread by choosing new targets. The difference between this stage and the target selection is simply from the point of view of the machine which the actions take place from. In target selection, a remotely infected host chooses the local host as a target. In the propagation stage, the infected local host is the one choosing a new target, using probes going out in outbound network traffic.

According to the aforementioned categorization, it is clear that the typical malicious script codes can exhibit many similar activities [20], [21]. By grouping their common actions, it can be concluded that malicious VBScript can perform one or more of the following basic functions:

- ***Registration***, allows the malicious code, to register itself on infected machines for future reference. Discovering the attack code during this stage is quite difficult, as the malicious code would restrict its activity to an unnoticeable level.
- ***Infection***, is the core of the malicious code, and contains the destructive part of the virus. Typically, this includes effects such as loss of data or sluggish system response.
- ***Replication*** is the method where a malicious code can survive by infecting new systems.

[13] It is a computer Worm, discovered in July 2001. It only infects windows machines running Microsoft IIS web servers. It exploits a bufferoverrun vulnerability which was by no means new when the worm started to spread.

According to the above discussion, malicious VBScript can be detected by extracting abnormal 'features' from malicious codes that deviate from normal behavior.

c. 'Features' of attack codes

In order to be able to group the different behavioral patterns of normal or attack codes, it is necessary to introduce a definition of the features that can be used to identify such patterns.

A feature can be defined as a particular *regular expression or element* that is scanned over a particular code. Each time the regular expression matches, the feature type statistics is set by applying different *encoding schemes*.

Accordingly, *a feature type* can be thought of as a set of:

a. *Regular expression or element:* which is a fragment or constituent of a larger unit (script code), it is composed of VBScript lexically related keywords, function, or methods that are combined together to form a specific pattern or feature.

b. *Encoding schemes:* are the statistics used to record a match found of a regular expression within the code. There are several encoding schemes used in the current problem such as:

   Repetition (count): it indicates the total number of times an element appears in normal or malicious codes

   Occurrence: it indicates the existence of an element within the code.

In order to illustrate this concept, the following example shows *a typical malicious feature* that uses MS Outlook to mass-mail other hosts. The worm uses distinct keywords to propagate via MS Outlook (its basic function is replication as stated above)[21]:

1. First step, it creates an object to enable using MS Outlook. In VBS this is done by calling the `'CreateObject'` method and putting `'Outlook.Application'` as a parameter. Hence the lexical word used to identify that segment of the feature is
   - `"Createobject(Outlook.Application)`

2. After creating an object equivalent to MS Outlook application, it is necessary to gain access to the mail system in order to be able to send emails. This is carried by getting the namespace for the Mail Application Programming Interface (MAPI). That step is typically carried using the following command
    - `"Getnamespace("MAPI")`

3. After opening MS Outlook and getting access to the mail system, the next step is to get the address list from the current user on the victim machine. This step usually involves recalls to the following properties
    - `"Addresslists"`
    - `"Addressentries.count"`

4. The final step in the replication feature is to create a mail message and send it to the identified address list. The mail has a subject line, body, and attached document, hence that segment of the replication feature would typically use any/all of the following properties
    - `"CreateItem"`
    - `"Subject"`
    - `"Body"`
    - `"Attachment"`
    - `"Send"`

4.3 Pattern Identification (feature extractor)

4.3.1 Application of Principal Component Analysis to the feature extraction problem

VBScript language is quite a simple language for programming. Yet despite its simplicity, the ability to get relations that combine a group of VBScript features can be computationally intensive. The regular expressions used in the current problem are so large and lies in the range of 243 elements of VBScript lexically related keywords,

functions, properties, or methods [44]. Accordingly, relations between patterns are hard to find in data of high dimensions. The larger the elements that constitute the data, the more difficult is identifying patterns by highlighting their similarities and differences.

As outlined earlier, there are several statistical multivariate techniques that are used to deduce the relations among a data set. In particular, PCA (see Appendix C) is used to reduce the dimensionality of data in which a large number of variables are interrelated. PCA accomplishes this by computing a smaller set of uncorrelated variables which best represent the original data. In this case, the use of Principal Component Analysis (PCA) [57], [58] can greatly help in reducing the dimensionality of the problem while keeping the significant "principal components" in the reduced data set. It is therefore desirable to extract measurements that are invariant or insensitive to the variations within each class. The process of extracting such measurements is labeled hereafter as "*feature extraction*". Thus, PCA is used as a feature extractor to extract patterns that will be used by the classifier, which follows the feature extractor, to differentiate between normal and malicious script.

4.3.2 Definition of Principal Component Analysis

As explained in chapter 3, Principal Component Analysis (PCA) [58] represents a classical statistical technique that is used to analyze the covariance structure of Multivariate statistical observations. The idea behind PCA is quite old. It roots back to Pearson [57] in 1901 as a methodology for fitting planes in the least-squares sense (like in linear regression). It was Hotelling (1933) [57] who proposed this technique for the purpose of analyzing the correlation structure between many random variables.

The purpose of PCA is to identify the dependence structure behind a Multivariate stochastic observation in order to obtain a compact description of it. So, PCA replaces the original highly correlated variables of a data set with a smaller number of less correlated variables called the *principal components*. These principal components are the key features of the random observation vector and in the IDS context can be used to differentiate between benign and malicious script [57], [58].

If the original data set of dimension *n* contains highly correlated variables, then PCA maps the data into a different space where each element represent a dimension of that

space, and by observing the relative sizes of the mapped-out data, only the higher (principal) *m* components of the data are selected that describes most of the data. The presence of fewer components (*m)* makes it easier to label each dimension with an intuitive meaning. The *n* observed variables are thus represented as functions of *m* latent variables called *factors*, where $m < n$ (and often $m << n$). Thus, such analysis has a great advantage of reducing the data dimensionality that might in many cases be of exponential order while maintaining the information content [57].

In a statistical context the number *n* is called the *superficial dimensionality* of the data, where *m* is called the *intrinsic dimensionality* of the data. The stronger the correlation between the observed variables is, the smaller the number of independent variables (m) that can adequately describe them [57].

4.3.3 Steps to perform PCA in feature extraction

As illustrated above, PCA [57] is a powerful tool for analyzing data; it is used to find patterns in data of high dimensions by reducing the number of dimensions without loss of information. The following represents the steps used by PCA to reduce the VBScript elements' dimensionality by identifying the correlation between them [58].

a. Data normalization.
b. Calculation of the covariance matrix and correlation matrix.
c. Calculation of the eigenvectors and eigenvalues of the covariance matrix.
d. Choosing the principal components.

A parser (feature extractor) was developed (using the C language) to extract the existence and occurrence of VBScript elements (attributes of a specific feature) in a code population in Appendix A. This operation is applied on multiple files (benign and malicious files) to develop the occurrence matrix of VBScript elements (regular expression). The parser does the following functions to help analyze and understand how relevant each feature under test is:

a. Data Normalization

Following Cunningham's approach [19], the problem of identifying features of common benign code was translated to testing the occurrence and repetition (stated above) of VBScript elements (regular expression) on the benign data set used for training the extractor.

Collection of statistics around regular expression (VBScript elements):

As stated above, the parser collects statistics (occurrence and repetition) around VBScript elements' within the code population under test. A file that contains all the VBScript elements is scanned over malicious and benign scripts. Each time the regular expression (VBScript elements) finds a match within the code, this match is recorded.

Mean and Variance calculation:

The feature extractor was designed to compute the mean[14] (in Appendix B) of the occurrence of VBScript elements for malicious and benign script.

The calculation of the mean value (the average).

$$m = \frac{1}{n}\sum_{k=1}^{n} X_k \ . \qquad (5)$$

Where μ indicates the mean of the set X in (5).

It also computes the variance[15] (in Appendix B) in order to show the spread of data from the mean. Then, the parser runs over both malicious and benign code

[14] The mean of a collection of numbers is their arithmetic average, computed by adding them up and dividing by their number.

[15] A measure of the average distance between each of a set of data points and their mean value; equal to the sum of the squares of the deviation from the mean value.

population to compute the Total Variance (TV), Variance for Clean (VC), and Variance for Malicious (VM).

- **Total Variance (TV):** is the computation of the variance for the code population that constitutes both normal and malicious scripts.
- **Variance for Clean (VC):** it is the calculation of the variance for normal scripts code population.
- **Variance for Malicious (VM):** it is the calculation of the variance for malicious scripts code population.

The Total Variance Ratio (R) is computed as in the following equation (8), where the objective is Total Variance Ratio (R) maximization. This could be achieved by dividing the Total Variance (TV) by the Variance for clean (VC) multiplied by $Weight_1$[16] and Variance for malicious (VM) multiplied by $Weight_2$[17]. Thus, the Total Variance (R) is maximized for both Normal and Malicious codes.

The Total Variance Ratio (R) = ($Weight_1$ * (TV/ VC) + $Weight_2$ * (TV/VM)). (8)

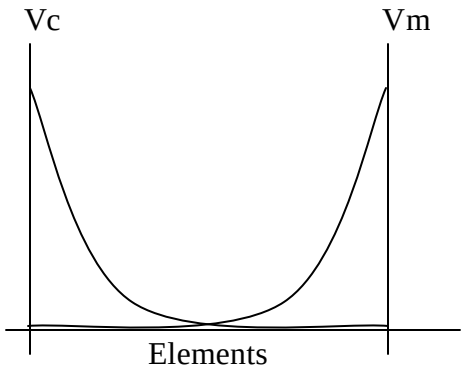


Figure 4. 2: Elements of Primary filter

[16] $Weight_1$ is the total number of normal scripts divided by total number of normal and malicious scripts' files.

[17] $Weight_2$ is the total number of malicious scripts divided by total number of normal and malicious scripts' files.

The Variance for clean (VC), and the Variance for Malicious (VM) in the above Figure 4. 2 could be used as *a Primary filter* to differentiate between benign and malicious scripts.

If Variance for clean (VC) of some VBScript elements records high value in clean scripts, while it records zero value in malicious scripts, or vice versa; thus, these VBScript elements are retained to be used as primary filter to differentiate between normal and malicious scripts.

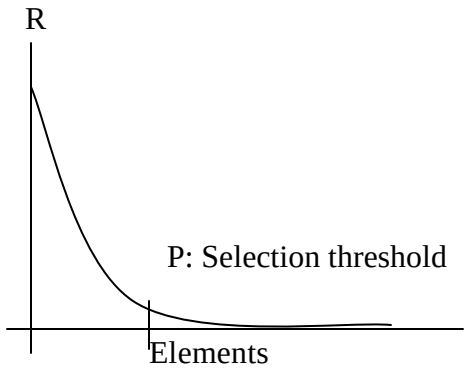


Figure 4. 3: Elements sorted according to their highest Total Variance Ratio (R)

After selecting the elements that will constitute the primary filter, the rest of VBScript elements are sorted in descending order according to their Total Variance Ratio (R). A threshold is set to select a number of elements P that have the highest Total Variance Ratio (R), and discard those with lowest values. Thus, the elements' size is reduced, while retaining the most information see Figure 4. 3.

The following step illustrates the calculation of the covariance matrix P × P on these reduced or selected number of VBScript elements P.

b. The Computation of the Covariance Matrix or the Correlation matrix:

The rotation of the existing axis is computed by getting the Covariance between the selected P elements. Thus, the covariance matrix (see Appendix B) is a square matrix of dimension equal to P.

The covariance Matrix or a *correlation matrix*[18] is always measured between 2 dimensions as described in the following equation (6):

$$COV(X,Y) = (1/n)\sum_{k=1}^{n}(X_k - \overline{X})(Y_k - \overline{Y})^T . \qquad (6)$$

Where $\overline{X}$ and $\overline{Y}$ are the mean of the set X and Y.

The correlation between a pair of variables is equivalent to the covariance divided by the product of the standard deviations of the two variables X and Y, as described in the following formula (7):

$$\text{Correlation matrix} = \frac{COV(X,Y)}{\partial_X \partial_Y} . \qquad (7)$$

The correlation between the elements in the correlation matrix can be categorized as follows:

- **Low correlation:** the correlation between the elements is less than 30%.
- **Medium correlation:** the correlation between the elements is in the band 30-70%.
- **And high correlation:** the correlation between the elements is higher than 70%.

Thus, if two elements are highly correlated (correlation between the elements higher than 70%) in the correlation matrix; so, one element of them is selected, the one with the highest Total Variance Ratio (R) in the computation of the eigenvalues and eigenvectors. If there is no correlation between the elements or the correlation is low, all the elements are taken into account to perform the computation of the eigenvalues and eigenvectors.

c. Computation of the Eigenvalues and Eigenvectors of the Covariance matrix:

In this step, Eigen-decomposition is calculated by finding the roots *eigenvalues and their corresponding eigenvectors*[19] (in Appendix B). Matlab is used to perform this

[18] The results of Correlation Matrix are the same as the Variance-Covariance Matrix because the correlation between a pair of variables is equivalent to the covariance divided by the product of the standard deviations of both variables.

computation of eigenvalues and eigenvectors. In general, once eigenvectors are computed from the covariance matrix, then they are sorted according to their highest eigenvalues in order to give the elements that constitute the components an order of significance. So, the decision can easily be made at this stage to ignore the components of lesser significance while keeping those of higher significance. The higher the eigenvalues, the more characteristic features of a script does the particular eigenvector describe [58], [64].

d. Choosing the principal components

The eigenvector with the highest eigenvalue is the principal component of the data set that points down the middle of the data. Elements with low eigenvalues can be omitted as they only describe a small part of the characteristic features of the script.

A feature vector (eigenfeatures) is formed by setting a threshold to select a number of elements (M') from the list of eigenvectors that have the highest eigenvalues. Thus, the elements' size is reduced and these elements, which are easily used as the main principal components, constitute *the secondary filter* to differentiate between normal and malicious script.

4.4 Attack Classifier

Once the feature vector (The M' significant eigenvectors of the covariance matrix P × P are chosen as those with the largest associated eigenvalues) is created, identification becomes a pattern recognition task.

Those *eigenfeatures or (eigen VBScript elements)* span an M' dimensional subspace of the original N VBScript elements space. This eigenfeature vector M' is used to find which of the two pre-defined classes (Normal or Malicious script) best describes the script.

The simplest method for determining which class provides the best description of an input script is to find *the Euclidean distance* [64] from the centre of both clean and malicious codes.

---

[19] Eigenvector is a vector which, when acted on by a particular linear transformation, produces a scalar multiple of the original vector. The scalar in question is called the eigenvalue corresponding to this eigenvector.

$$D_i^n = \sum_{j=1}^{m} \left\| a_{ij} - \bar{a}_j^n \right\|^2 \qquad (9)$$

Where in (9) $\bar{a}_j^n$ is the mean (center) of each class Normal or Malicious script, and a_{ij} represents the script under test.

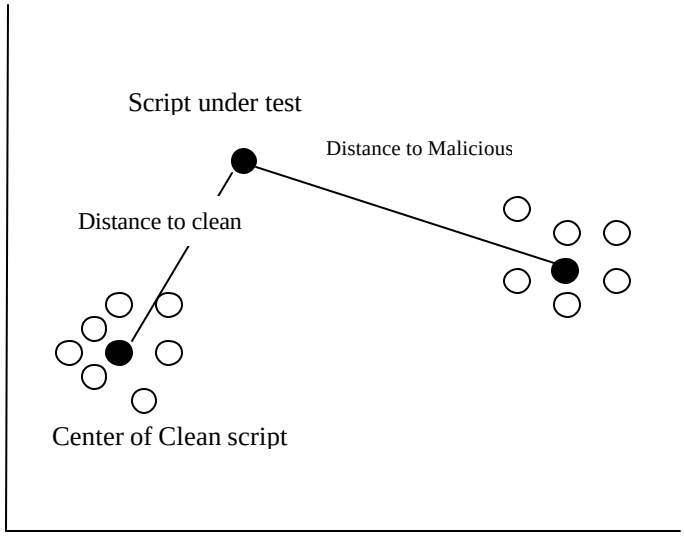


Figure 4. 4: The Computation of the Euclidean Distance from the center of both clean and malicious codes.

Euclidean Distance [52], [64] as in Figure 4. 4 is a straight line distance used as a measure of similarity in the nearest neighbor method. Classification based on this distance measure is direct and simple where the mean values of the elements in malicious or normal script are used as class centers. A script is classified as belonging to malicious class, if the Distance to Malicious center is less than the Distance to Normal center. Otherwise, it belongs to normal class. This means that when the minimum Euclidean Distance D_i^n is below some chosen threshold (h) from the center of benign data population, a script can be categorized as benign script.

In testing the classifier, if a test sample is benign yet the measured Euclidean distance is higher than the threshold, the case represents a false positive case. On the other hand if the test sample is for a malicious script, and the measured distance is less than the threshold, the case represents a false negative [8].

It is clear from the above that the selection of that threshold has a key impact on the accuracy of the classifier. Selecting a large threshold would lead to the generation of a high false negative rate (missing detection), yet selecting a tight threshold would lead to the generation of a high false positive rate (wrong detection).

4.5 Results Analysis & Validation

As stated earlier, the 2 main objectives of the work are:

- To isolate features common to the malicious attacks, and
- To propose an implementation for an attack classifier that would allow for using these features in order to efficiently detect VBScript malicious attack.

4.5.1 Efficiency and performance measures

First in doing so, there is a need of setting measurements to quantify *the system's efficiency*. The following are measurements used in our experiments [8]:

1. There are four measures used, such as *true positives, true negatives, false positives, and false negatives.*

- A true positive is a malicious example that is correctly tagged as malicious.
- A true negative is a benign example that is correctly classified as benign.
- A false positive is a benign program that has been mislabeled by an algorithm as malicious.
- While, a false negative is a malicious executable that has been misclassified as a benign program.

2. *The detection rate* is the number of malicious binaries correctly classified divided by the total number of malicious programs tested.

For the '*feature*', efficiency would be defined through the relation between the percentage of feature hit to the population code (normal and malicious code), in other words "how many abnormal feature elements are present in how much of the code population". According to that measure *an efficient feature* (a good representative of malicious feature) will score high when applied over the malicious code population (and alternatively will score low when applied over the normal code population). This

efficiency measure will support demonstrating the validity of the *1st hypothesis*. The parsing system stated earlier allows for testing the efficiency of the isolated features.

For the *'attack classifier'*, efficiency would be defined as establishing a high rate of attack detection with a low rate of false positive alarms. This measure will support demonstrating the *2nd hypothesis*. It will be depicted using ROC curves (Receiver Operating Characteristic) [27], which are widely used to measure the effectiveness of an intrusion detection system by plotting the likelihood that an intrusion is correctly detected against the likelihood that a non-intrusion is misclassified (false positives). This will allow for judging on the effectiveness of the system.

Both *efficiency measures* allow for validating the assumptions (hypotheses) upon which the work is built.

4.5.2 Experiments for training and testing

Two types of experiments will be used to train and test the system:

1. Experiment over unseen VBScript: where *a cross validation*[20] [35] is used to experiment the proposed system in estimating the results of detecting new malicious VBScripts. *K fold cross validation* [35] is the standard method to estimate the performance of predictions over unseen data, which means that the data is divided into k subsets of approximately equal size. The system will be trained k times, each time leaving out one of the subsets from training, but using only the omitted set to compute whatever criterion.

   The system will use *five fold cross validation*, where the population code (normal and malicious scripts) will be partitioned into five equal size sets. Four of these sets will be used for training the system and then the remaining set will be used for evaluation. This process will be repeated five times by leaving out a different set for testing every time. The results of these tests will be averaged to obtain a measure of how the system performs in detecting new malicious VBScript. This

[20] Cross validation is a method aiming at estimating the error of a hypothesis, generated by a concept learning algorithm (classifier).

method is envisaged to give a reliable measure of the new system's accuracy on unseen data.

2. Experiment over known VBScript: this experiment will evaluate the performance of the system on detecting known VBScripts. For this task, the system will generate detection for the entire training set of data (normal and malicious code population). This means that the same data set will be used for training and for testing the system.

4.6 Conclusion

The work introduced in this chapter is an extension on a similar research line followed by Cunningham [19]. The proposed system is an anomaly host based IDS that provides protection against web attacks from malicious VBScripts.

It is based on two important hypotheses: the 1st hypothesis is that the majority of malicious codes exhibit different behaviors in comparison to normal codes, at the same time whether the codes within each group share common patterns. While the 2nd hypothesis assumes that the code patterns exhibited in malicious VBScripts are unique to attack codes and record very low presence in normal VBScripts.

To prove the validity of the hypotheses proposed, a large data set was collected for both normal and malicious scripts in order to study the nature of normal VBScripts and detect how malicious script deviates from the normal profile.

Moreover, the proposed technique comprises two significant components: *a feature extractor (pattern identification) and an attack classifier*.

The proposed feature extractor is based on a Multivariate Statistical technique, Principal Component Analysis (PCA) [57] that extracts common patterns to distinguish between normal and malicious script. PCA analysis is carried on the proposed problem to reduce the dimensionality of the problem while keeping the significant "principal components" in the reduced data set. Thus, PCA highlights the key differences between patterns of benign and malicious scripts.

An attack classifier, which follows the feature extractor, uses the Euclidean distance [64] to learn the best combination of feature elements in order to categorize the incoming script into normal or malicious code.

The following chapter presents an analysis of the results of the proposed system in order to demonstrate the validity of the hypotheses proposed.

Chapter 5: Results Analysis and Validation

5.1 Introduction

As outlined in chapter 4, the development of IDS passed by three main stages: *malicious code study, pattern identification and development of the attack classifier*. This chapter will go through the detailed steps taken in order to generate the results in each stage of the IDS development and analyze them. A key outcome as well is to verify the initial assumptions (hypothesis) on which the work is built. Finally, the methodology used in testing with the results will be presented and discussed.

5.2 Input Data Analysis

To test the proposed system, the first stage of the work focused on building a collection of both normal (non-malicious) and malicious codes. Malicious script source codes (as stated in the previous chapter) were gathered from various sources, and it was possible to collect a population of VBS attacks representing exactly 283 malicious VBScript attacks (see Appendix D) known (on McAfee and Symantec databases).

The study of the malicious code revealed that the attack source codes usually perform one of the following actions:

- Access the registry
- Access the user's mailing list
- Launch tasks
- Create users and groups
- Change user's permissions
- Change and create files
- Change the directory structure

It was thus essential to build the benign code population that covers both complete applications as well as short task-specific scripts. The collected population of benign scripts is in the actual of 1113 normal scripts in Appendix D, as described in Table 5. 1 below

In general, the benign data set was collected to maintain the diversity of the following source code properties such as:

- Nature of task performed: the benign scripts were collected to cover a wide range of application including the actions that were identified earlier during the study of the malicious code.
- Source: the benign data was collected from a variety of sources in order to make sure that it covers a wide range of programming styles.

Table 5. 1: The population of collected benign and malicious scripts

	Malicious	Benign
Script files	283	1113

The feature elements of the VBScript language are the various VBScript related keywords such as methods, properties, and functions available in the language. They are exactly 243 elements in total. The following Table 5. 2 shows the feature elements of VBScript used in the analysis [69], [70].

Table 5. 2: Total VBScript elements or VBScript lexically related keywords used in the feature list [69], [70].

Abs(	IsArray(	StrReverse(	RegRead(	Named	.Environment(
Array(	IsDate(	Tan(	RegWrite(	Number	.Error
Asc(	IsEmpty(	Time	Remove(	Path	.ExitCode
Atn(	IsNull(	Timer	RemoveNetworkDrive(	ProcessID	.FullName
CBool(	IsNumeric(	TimeSerial(	RemovePrinterConnection(	RelativePath	.Hotkey =
CByte(	IsObject(	TimeValue(	Run(	ScriptFullName	.IconLocation =
CCur(	Join(	TypeName(	Save	ScriptName	.Interactive
CDate(	LBound(	UBound(	SendKeys(	Source	.Item(
CDbl(	LCase(	UCase(	SetDefaultPrinter(	SourceText	.length
Chr(	Left(	Unescape(	ShowUsage	SpecialFolders(	.Line
CInt(	Len(	VarType(	Sign (	Status	.Name
CLng(	LoadPicture(	Weekday(	SignFile (	StdErr	.Named
Cos(	Log(	WeekdayName(	Skip(	StdIn	.Number
CreateObject(	LTrim(	Year(	SkipLine	StdOut	.Path

CSng(	RTrim(	AddPrinterConnection(	Sleep(	TargetPath	.ProcessID
CStr(	Trim(	AddWindowsPrinterConnection(	Terminate	Unnamed	.RelativePath
Date	Mid(	AppActivate	Verify (	UserDomain	.ScriptFullName
DateAdd(	Minute(	Close	VerifyFile (	UserName	.ScriptName
DateDiff(	Month(	ConnectObject(	Write(	Version	.Source
DatePart(	MonthName(	Count	WriteBlankLines(	WindowStyle	.SourceText
DateSerial(	MsgBox(	CreateObject(	WriteLine(	WorkingDirectory	.SpecialFolders(
DateValue(	Now	CreateScript(	Arguments	HKEY_CURRENT_USER	.Status
Day(	Oct(	CreateShortcut(	AtEndOfLine	HKCU	.StdErr
Escape(	Replace(	DisconnectObject(	AtEndOfStream	HKEY_LOCAL_MACHINE	.StdIn
Eval(	RGB(	echo	BuildVersion	HKLM	.StdOut
Exp(	Right(	EnumNetworkDrives	Character	HKEY_CLASSES_ROOT	.TargetPath
Filter(	Rnd(	EnumPrinterConnections	Column	HKCR	.Unnamed
FormatCurrency(	Round(	Exec(	ComputerName	HKEY_USERS	.UserDomain
FormatDateTime(	ScriptEngine	Execute	CurrentDirectory	HKEY_CURRENT_CONFIG	.UserName
FormatNumber(	ScriptEngineBuildVersion	Exists(	Description	WScript	.Version
FormatPercent(	ScriptEngineMajorVersion	ExpandEnvironmentStrings(	Environment(	("WScript.Network")	.WindowStyle
GetLocale(	ScriptEngineMinorVersion	GetObject(	Error	("WScript.Shell")	.WorkingDirectory
GetObject(	Second(	getResource(	ExitCode	.Arguments	.PrivateProfileString(
GetRef(	SetLocale(	LogEvent(	FullName	.AtEndOfLine	.ProfileString(
Hex(	Sgn(	MapNetworkDrive(	Hotkey	.AtEndOfStream	Clear
Hour(	Sin(	Popup(	IconLocation	.BuildVersion	Raise
InputBox(	Space(	Quit(	Interactive	.Character	replace
InStr(	Split(	Read(	Item(	.Column	Test(
InStrRev(	Sqr(	ReadAll	length	.ComputerName	
Int(	StrComp(	ReadLine	Line	.CurrentDirectory	
Fix(	String(	RegDelete (	Name	.Description	

5.3 Parser design and operation

In order to be able to test the presence of specific features in the code, it was essential to develop a tool that would help extracting the features from the benign data set described above. A program was developed (hereafter called 'the parser') see Appendix A. A parser was developed in C language due to its flexibility in file manipulation; it performs the following tasks:

1. Reading a parameter file that includes the features whose presence is required to be tested, as well as the location of the source codes.
2. Looping over the source codes and tests the occurrence of each of the elements in the features' list.

3. Building a matrix containing the repetition of each element in the features' list (the rows represent the source codes and the columns represent the features' elements).
4. Building another matrix containing the occurrence (if the repetition is higher than one, the occurrence flag is set to one, otherwise it is set to zero).
5. Performing correlation analysis between the elements.
6. Writing the results of the analysis onto an output file showing the parsing as well as the statistical analysis.

5.4 Multivariate Statistical Analysis: PCA

5.4.1 Data Normalization

The feature extractor (parser) was designed to compute the mean of the occurrence of VBScript elements for malicious and benign script. It also computes the variance in order to show the spread of the data from the mean. The Variance for clean (VC), the Variance for Malicious (VM), and the total variance Ratio (R) were computed for clean and malicious scripts (combined code population).

After applying the Parser on the benign data set using the feature elements outlined above, the next step is to narrow down the features selection from the current set of 243 elements to a reduced and more descriptive set of feature elements that best describe the benign behavior.

This is done over a series of steps that filters the number of elements in order to reach a reduced set of high significance (principal components). These steps are,

a. Elimination of low usage elements: By computing the total variance Ratio R (over the combined code population). The elements corresponding to low R values represent elements which are rarely used see Table 5. 3 In the following analysis, the cutoff is taken as R = 0.

Table 5. 3: Low Usage Elements

Atn(	oct(	createscript(	skip(
Ccur(	scriptenginebuildversion	disconnectobject(	.currentdirectory
Dateadd(	setlocale(	enumprinterconnections	.exitcode

datepart(	tan(	getresource(	.hotkey =
dateserial(	timeserial(	logevent(	.relativepath
formatcurrency(	weekday(	mapnetworkdrive(	.sourcetext
formatdatetime(	weekdayname(	removenetworkdrive(	.stderr
formatpercent(	addprinterconnection(	removeprinterconnection(	.profilestring(
getref(	addwindowsprinterconnection(	sendkeys(	
Rtrim(	connectobject(	signfile (	

This step managed to reduce 18% (43 elements) of the initial elements set. The eliminated elements include for example mathematical functions (e.g. Atn, Tan, ..) which are not commonly used in VBScript (being a non-technical script language). It also contains string manipulation elements (e.g. formatpercent, ..) which are usually handled by the html page.

Two cascaded filters such *as primary and secondary filter*s are used by the attack classifier as described below:

b. Primary filter:

1. Selection of elements with high VC: After computing the variance over the clean code set, the elements that score high over the benign code while maintaining a low variance over the malicious code, is selected as the first set of elements to be used in the classifier. In the following analysis, elements with VM = 0 with high VC are selected.

In this step approximately (22 %) 53 of VBScript elements are retained to be used in the *primary filter* to differentiate between normal and malicious scripts illustrate in Table 5. 4 below.

Table 5. 4: Elements used in the primary filter to detect normal scripts.

array(	.scriptname	cdbl(	.unnamed
quit(	lbound(	getlocale(	cos(
.version	year(	regdelete(	csng(

isnull(	.column	fix(	eval(
showusage	.userdomain	strcomp(	loadpicture(
cbool(	isdate(	sleep(	monthname(
.stdout	typename(	test(	sgn(
log(	terminate	formatnumber(	skipline
isempty(	.interactive	scriptengineminorversion	.atendofline
verify (	datevalue(	sqr(	.buildversion
cint(	filter(	vartype(	.character
isobject(	timevalue(	setdefaultprinter(	
remove(	.processid	verifyfile (	
.error	raise	.stdin	

2. Selection of elements with high VM: elements that have high variance over malicious code set and low variance over clean code set. It can be used as a source of differentiation in the primary filter by reciprocating its presence, i.e. by detecting its 'absence'. This process led to the selection of 5 elements.

Table 5. 5: Elements used in the primary filter to detect malicious scripts.

.privateprofilestring(	appactivate	regwrite(	regedit(
Cbyte(			

After selecting the elements that will constitute the primary filter, the rest of VBScript elements are sorted in descending order according to their Total Variance Ratio (R) as shown in the following Figure 5. 1.

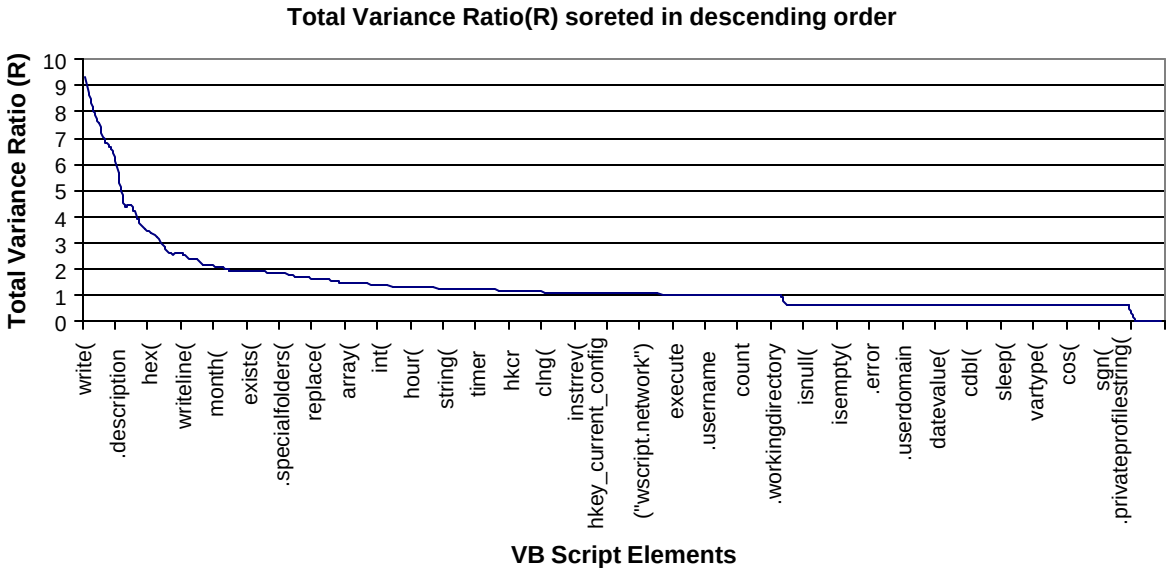


Figure 5. 1 : Total Variance Ratio(R)

c. The R factor test

In this step, a threshold is set to select a number of elements P that have the highest Total Variance Ratio (R), and discard those with lowest values. Thus, 20 % of the elements that have the highest Total Variance Ratio (R) are selected, so the elements' size is reduced while retaining the most information as illustrated in Figure 5. 2, and Table 5. 6.

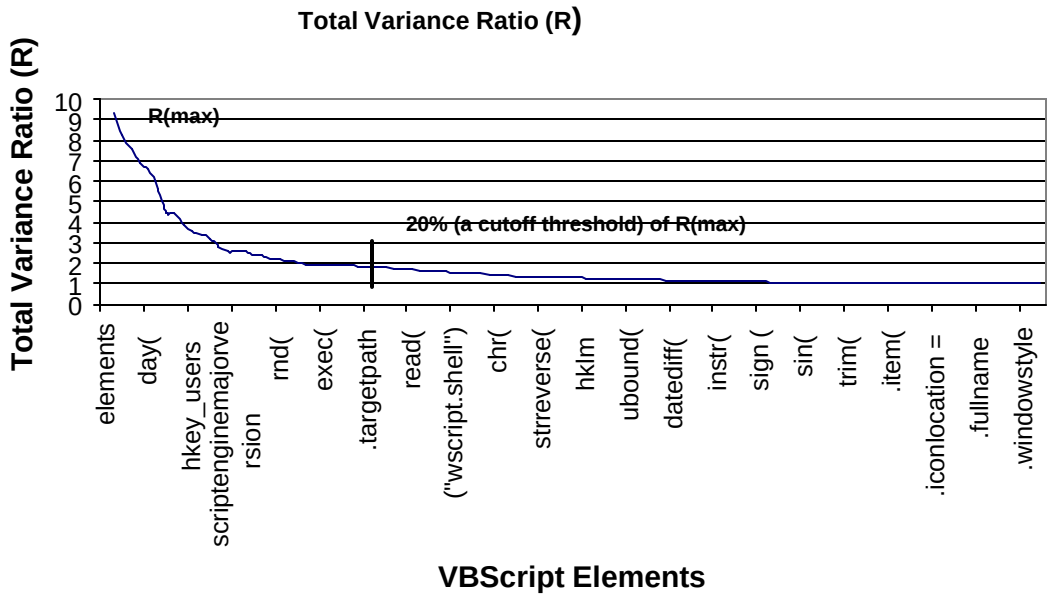


Figure 5. 2: The R factor test.

Table 5. 6: The selection of 20% (a cutoff threshold) of the elements that have the highest Total Variance Ratio (R).

.number	.scriptfullname	.arguments	write(
cstr(	writeline(	.specialfolders(	replace(
array(	scriptengine	readall	wscript
hkey_users	hkey_current_user	clng(	ltrim(
Second(	hex(	rgb(	hkey_local_machine
hkey_classes_root	echo	asc(	writeblanklines(
.description	rnd(	getobject(	

The above group of keywords was found to strongly correlate in VBScript attacks. By analyzing these keywords, two distinct subcategories can be identified:

Registry related keywords (hkey_users, hkey_classes_root, hkey_current_user, hkey_local_machine), and these are mostly used in attacks in order to register the virus on the infected machine, which is generally done by installing a registry key.

File structure related keywords (.specialfolders(), write(), writeline(), replace(), writeblanklines( ), readall), and these are VBScript keywords which are used to alter the file structure.

It is worth noting that the presence of any of the above elements, individually, does not provide enough evidence for suspecting an attack, yet it is the combination of all/most of the above at the measured correlation factors which provides such evidence.

5.4.2 The Computation of the Covariance Matrix (or the Correlation matrix)

As stated in the previous chapter, the rotation of the existing axis is computed by getting the Covariance between the selected P elements (reduced VBScript elements). Therefore, the covariance matrix is a square matrix (symmetric matrix) of dimension equal to P; the number of selected elements is P = 27, the covariance matrix will be 27 × 27.

The following Figure 5. 3, Figure 5. 4, Figure 5. 5 represent a color map of the calculated covariance matrix P × P over malicious, normal, and combined (both normal and malicious) scripts.

Figure 5. 3: Color map of Elements' Correlation for Malicious codes

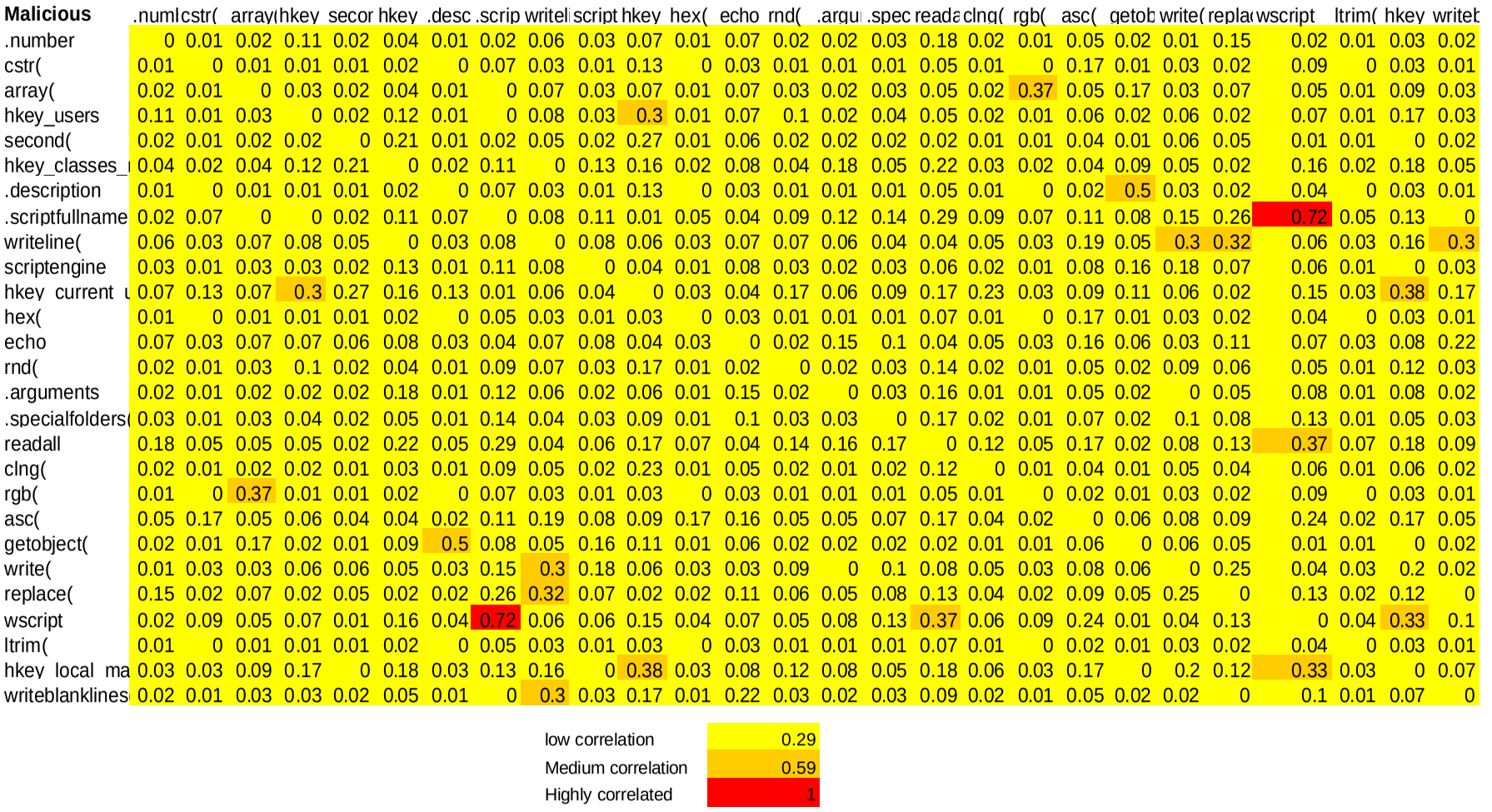


Figure 5. 4: Color map of Elements' Correlation for Normal codes

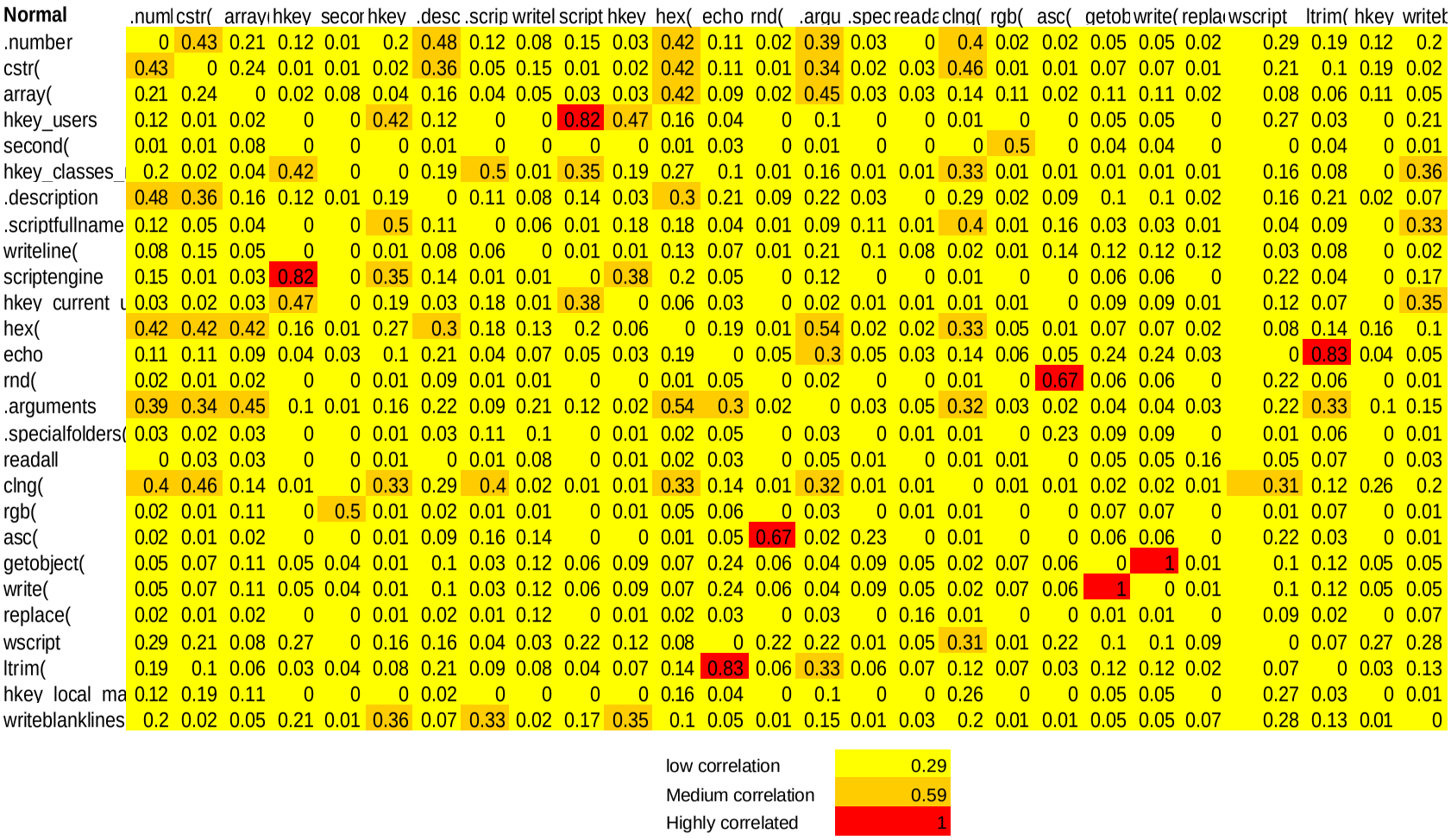
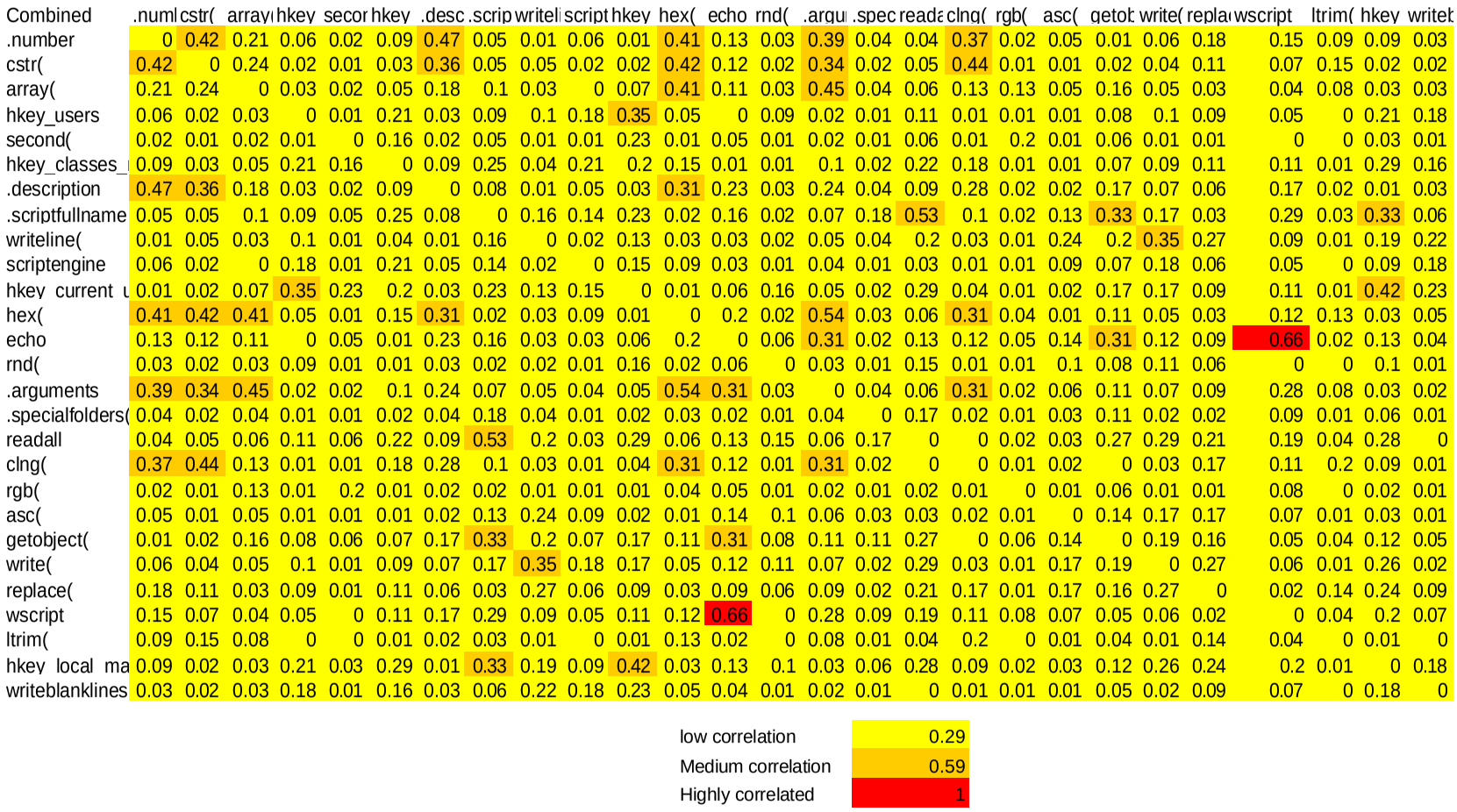


Figure 5. 5: Color map of Elements' Correlation for Combined codes



After the computation of the correlation matrix for *malicious, normal, and combined scripts (normal and malicious scripts together),* the correlation between the elements in the three matrices is inconsistent; this implies that all the elements are needed to be taken into account (i.e. there is no correlation between them) to perform the eigenvalues and eigenvectors computation.

5.4.3 Eigenvectors and Eigenvalues

Matlab is used to perform the computation of eigenvalues and eigenvectors. In general, once the eigenvectors are computed from the covariance matrix, and then they are sorted according to their highest eigenvalues to give the elements that constitute the components an order of significance. So, the decision can easily be made at this stage to ignore the components of lesser significance while keeping those of higher significance, see Table 5. 7.

Table 5. 7: Elements sorted in descending order according to their eigenvalues

Write	3.70
Readall	3.35
.scriptfullname	1.92
asc(	1.54
day(	1.44
.description	1.24
getobject(	1.20
hkey_current_user	1.10
hkey_users	1.07
hex(	1.04
second(	1.00
cstr(	0.94
minute(	0.87
scriptenginemajorversion	0.80
writeline(	0.78
scriptengine	0.77
.line	0.70
rnd(	0.69
month(	0.61
Space(	0.53
escape(	0.49
unescape(	0.48

exec(	0.46
Exists(	0.41
computername	0.37
.arguments	0.30
createshortcut	0.18
targetpath	0.00

5.4.4 Selection of Principal Components

The resulting principal components were ordered according to their highest eigenvalues and several trials were taken in order to select the optimal number of principal components. The selection was bounded by the fact that selecting a high number of components will increase the system accuracy yet on the expense of complicating the calculations. On the other side, reducing the number of principal components on which the system is build would simplify its design yet would sacrifice the quality of the detection.

Several trials led to selecting 70% (as a cutoff threshold) to select the principal components. This choice led the selection of 20 elements representing the principal components of the system, as shown in Table 5. 8 below, these principal components, constitute *the secondary filter* to differentiate between normal and malicious scripts without sacrificing the system accuracy such as the high detection rate, and low rate of false alarms.

Table 5. 8: The feature vectors (the principal components) that constitute the secondary filter:

Write
Readall
.scriptfullname
asc(
day(
.description
getobject(
hkey_current_user
hkey_users
hex(
second(
cstr(
minute(
scriptenginemajorversion
writeline(
scriptengine
.line
rnd(
month(
Space(

These keywords can be identified in two distinct subcategories inVBScript attacks such as:

Registry related keywords (hkey_users, hkey_current_user) which are mostly used in attacks to register the virus on the infected machine

File structure related keywords (getobject, write(), writeline(), readall) which are used to alter the file structure

The presence of all these elements in a script provides enough evidence for suspecting an attack.

Figure 5. 6 illustrates this by plotting the first two principal components that have higher significance. Principal component 1 (Write) on the X axis against principal component 2 (readall) on the Y axis for normal and malicious scripts. These two principal components can easily identify and separate malicious from normal scripts.

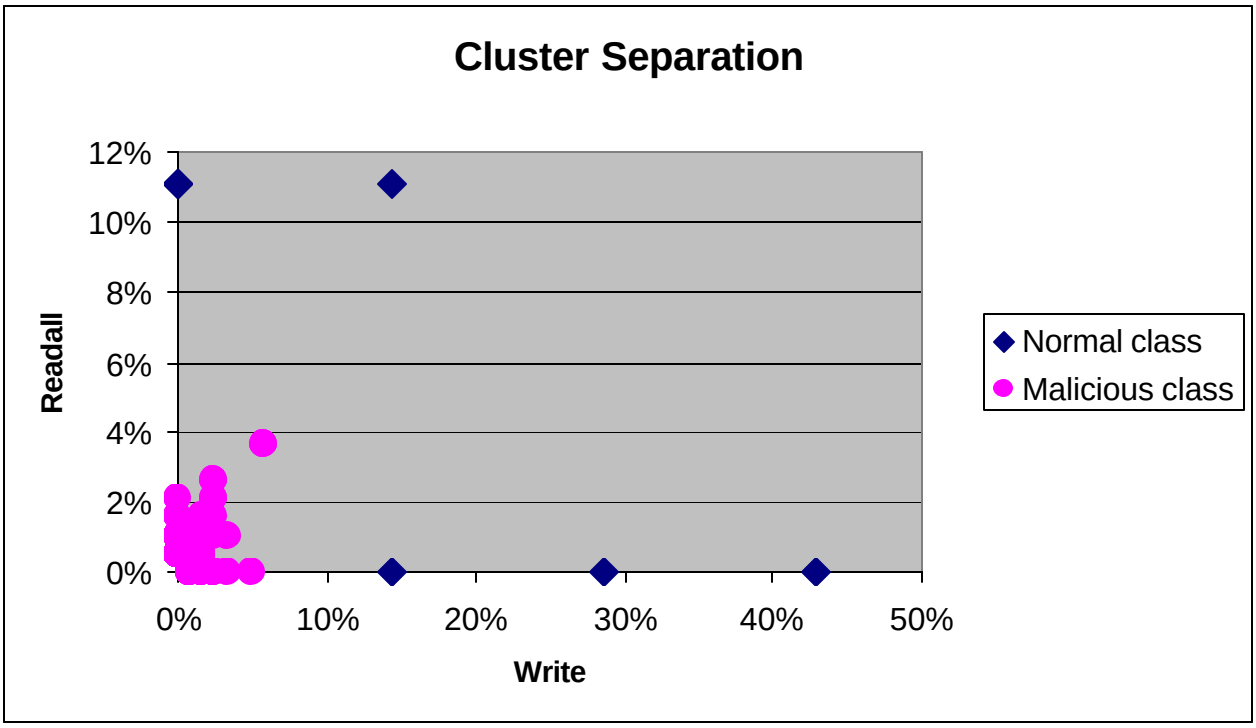


Figure 5. 6: The relation between Principal Component 1 (Write) and Principal Component 2 (Readall) that separates malicious from normal scripts.

5.5 Testing & Results

Detection will be based on Euclidean distance measured from the center of benign code (the mean of the class of normal script). Two detection methods were assumed based on whether the center of malicious code is used or not.

1. Detection Method 1: A code is malicious if the distance from the center of benign code is greater than the distance from the center of malicious code, i.e. the code is closer to the center of the malicious code than to the benign. In this detection method, the center of malicious is maintained to reflect the updated attacks. See Figure 5. 7.

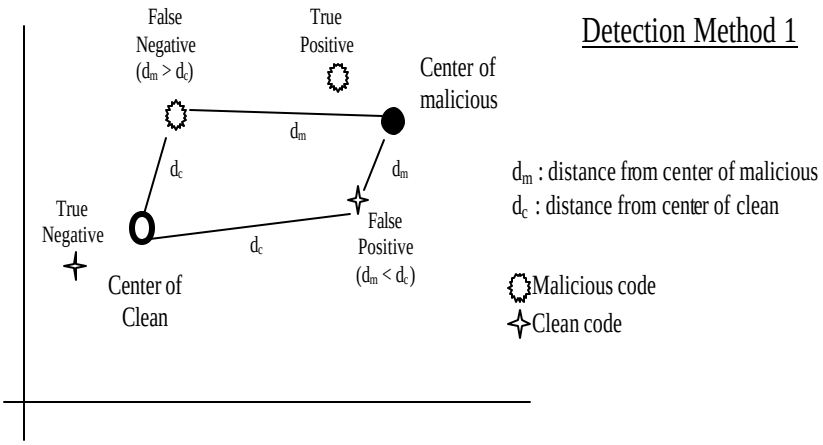


Figure 5. 7: Detection Method 1.

2. Detection Method 2: In this detection method, if the distance from the center of benign code exceeds a certain threshold 'h', the script is assumed to be malicious. The center of malicious is not used and the system depends wholly on maintaining a clean code base. See Figure 5. 8.

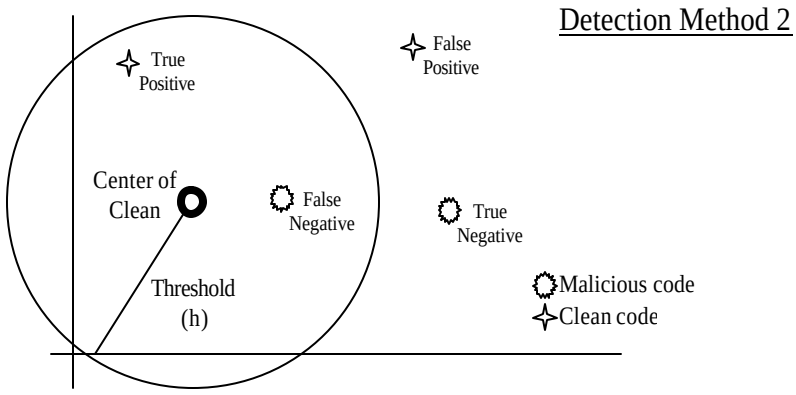


Figure 5. 8: Detection Method 2.

The first detection method offers a higher detection performance because it uses both the center of malicious and center of normal in the detection; while the second method offers the flexibility of changing the detection threshold and thus balancing the false positives

and false negatives. This is due to the fact that choosing a bigger threshold from the center of clean (mean of the clean class) will result in low false positives and high false negatives and choosing a small threshold from the center of clean will result in high false positives and low false negatives.

In order to test the performance of the detection, the following performance measures are used

1. True Positives (TP): number of malicious sources correctly classified as malicious
2. True Negatives (TN): number of benign sources correctly classified as benign
3. False Positives (FP): number of benign sources falsely classified as malicious
4. False Negatives (FN): number of malicious sources falsely classified as benign
5. Detection rate (DR): percentage of number of malicious sources correctly detected over all malicious sources (TP/(TP+FN))
6. False Positive rate (FPR): percentage of number of benign sources falsely detected over all benign code (FP/(TN+FP))
7. False Negative Rate (FNR): percentage of malicious code sources falsely undetected over all malicious code (FN/(TP+FN))
8. Overall Accuracy (OA): percentage of number of correctly detected sources (both benign and malicious) over all available sources (benign and malicious), (TP+TN)/(TP+TN+FP+FN)

5.5.1 Experiment with known data

The first experiment is to use the same data set for training and testing. The following Table 5. 9 summarizes the results as tested on both detection methods. For the second detection method, three threshold points were selected in order to illustrate the difference,

	Detection Method 1	Detection Method 2 (h=1)	Detection Method 2 (h=2)	Detection Method 2 (h=3)
FPR	2%	30%	12%	5%
FNR	26%	14%	29%	47%
DR	74%	86%	71%	53%
OA	93%	73%	85%	86%

Table 5. 9: Summary of experiment with known data

It is clear from the results that by using the center of malicious code, in addition to the normal profile in detection method 1, the false positive rate is minimized and the overall system accuracy is improved. So, the false positive rate was 2%, detection rate was 74%, and overall accuracy was 93%.

In order to illustrate the impact of choosing different threshold points, the following graph in Figure 5. 9 will show the variation of different key indicators such as false negative rate, false positive rate, and the overall accuracy against selecting different thresholds in detection method 2.

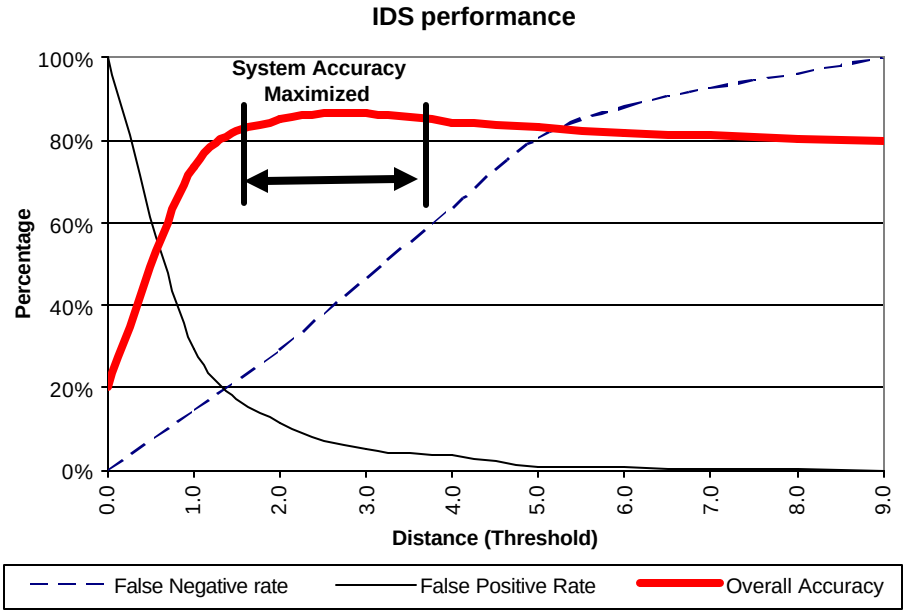


Figure 5. 9: IDS Performance.

Figure 5. 9 shows that in order to maximize the overall system accuracy, the attack classifier should be operated at a level of h in the range 1< h< 3. Different operating points result in different false positive and false negative rates. Increasing h towards the higher end of the operating range will result in minimizing the false positive rate, while selecting a low h will result in minimizing the false negative rate.

Figure 5. 10 depicts the relation between false positive rate and false negative rate in detection method 2 while varying the threshold 1< h< 3.

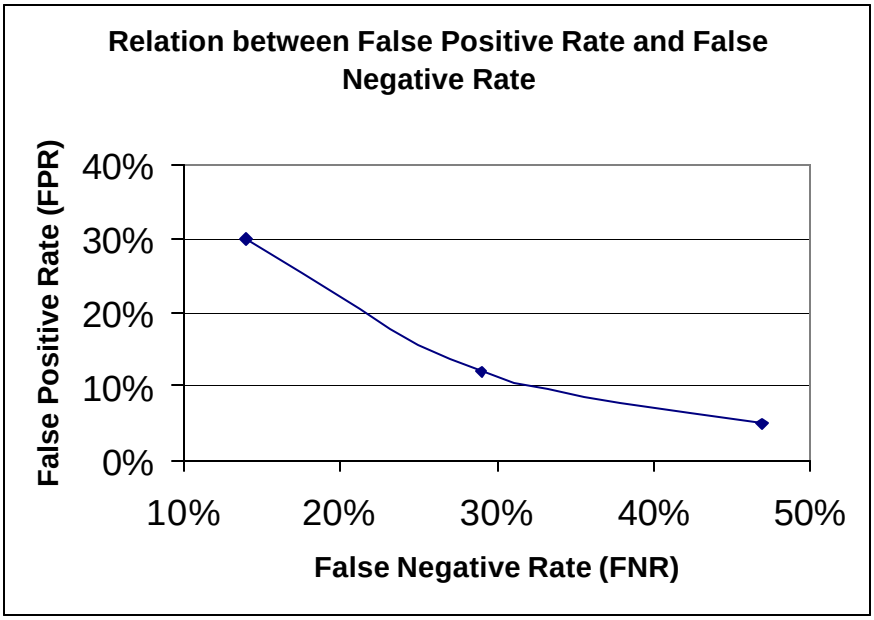


Figure 5. 10: Relation between False Positive Rate and False Negative in Detection Method 2.

The results of the second detection method (see Table 5. 9) show that at 12% false alarm rate, the detection rate was 75%. These results were compared with the results of Cunningham [19] who proposes a neural-network system to classify the nature of the attack. Although neural networks give better results this is due to the use of various examples during the learning phase. It was found that in C detector at a similar false alarm rate, the detection rate achieved was in excess of 90%, while the Shell detector yielded a detection rate similar to the proposed system 80%.

The results of both the proposed system and the shell detector give approximately similar results; the reason for this is that script languages are less structured, hence less predictable, than in structured language such as C.

The following curve in Figure 5. 11 shows the ROC curve (Receiver Operating Characteristic) [27] which is an aggregate of the probability of false alarms and the probability of detection measurements. The X axis shows the percentage of detected

attacks (DR) against the false alarm percentage. It is used to illustrate the results of the second method of detection applied over the seen data.

Comparing the ROC curve of the proposed system with the ROC curve for the (C detector, and Shell detector) in Cunningham [19], it was found that the ROC curve for the C detector outperforms both the proposed system and the Shell detector while the ROC curve for the proposed system is similar to the ROC curve of the Shell detector

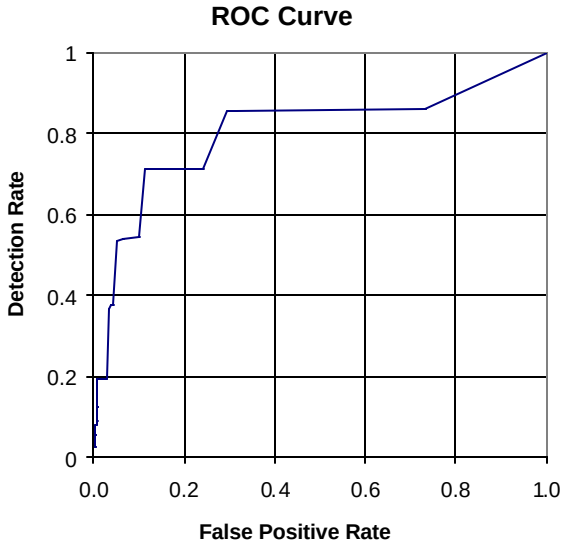


Figure 5. 11: ROC curve using the second method of detection applied over the seen data

5.5.2 Cross validation over unseen data

The following step is to test the system over 5-folds cross validation test. The purpose of the test is to highlight the impact of using the system in a real-life scenario, where the incoming data to the classifier will be unseen, and the normal profile (as well as the malicious center) is pre-calculated over a fixed set of data.

The experiment studies the impact of training the system over 80% of the original data set. The outcome of training is to calculate a new mean (center for both the clean and the malicious code) and then to use the new calculated profiles to test the incoming data.

The following Table 5. 10 summarizes the results tested by the system for the 5 folds cross validation. It is evident from the results that the average indicators (False Positive

Rate, False Negative Rate, Overall Accuracy, and Detection Rate) don't deviate significantly from the first experiment over seen data; it is consistent as well with the previous experiment. The second method of detection (based only on a threshold) still shows less attractive results in comparison to the first detection method. The first detection method offers a higher detection rate because it uses the center of malicious during the classification. Thus, a script can be easily classified as a malicious or normal depending on its distance from the center of benign and the center of malicious codes.

Detection Method 1		Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
	FPR	3%	3%	3%	3%	2%
	FNR	25%	21%	22%	25%	25%
	DR	75%	79%	78%	75%	75%
	OA	93%	93%	93%	92%	93%
Detection Method 2		Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
h = 1	FPR	30%	30%	30%	28%	28%
	FNR	14%	14%	14%	14%	14%
	DR	86%	86%	86%	86%	86%
	OA	73%	73%	73%	75%	75%
h = 2	FPR	12%	12%	12%	12%	11%
	FNR	29%	29%	29%	31%	29%
	DR	71%	71%	71%	69%	71%
	OA	85%	85%	85%	85%	85%
h = 3	FPR	5%	5%	5%	4%	5%
	FNR	47%	47%	47%	58%	47%
	DR	53%	53%	53%	42%	53%
	OA	86%	86%	86%	85%	87%

5 folds Average	Detection Method 1	Detection Method 2 (h =1)	Detection Method 2 (h =2)	Detection Method 2 (h =3)
FPR	3%	29%	11%	5%
FNR	24%	14%	29%	49%
DR	76%	86%	71%	51%
OA	93%	74%	85%	86%

Table 5. 10: Summary of cross validation results.

The following graph in Figure 5. 12 shows the ROC curve for the 5 folds cross validation. It is noticed also that the curve is largely similar to the ROC curve generated from the experiment over known data.

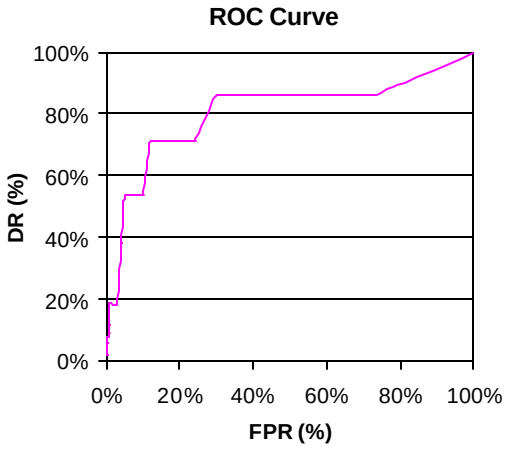


Figure 5. 12: ROC Curve for the 5 folds Cross Validation.

Chapter 6: Conclusion and Future Work

6.1 Research Summary

The need for dynamic content, feature-rich Web sites has lead to the adoption of various scripting languages. JavaScript, and VBScript are perhaps two of the commonly used ones nowadays. Designed with flexibility in mind, the adoption of these technologies within popular Internet applications and operating systems has led to the widespread problem of viruses, Trojan horses, worms, and script attacks.

Web Script codes are transmitted as source codes over the Internet and are automatically executed on any target machine. Accordingly they are hardware independent. This feature is among other features of VBScripts that make it receiving increasing focus as a backdoor for attacking computers by threatening the user's computer through sending itself in e-mail attachment or through browsing an infected site. What also led to the widespread of malicious attacks in VBScript is that malicious script can restrict its activity to unnoticeable levels by changing the windows registry, causing loss of data, and then spreading itself to infect other systems.

Commercial virus scanners are *the first line of defense* that protect against these malicious mobile codes. They can effectively detect known attacks, but are not able to effectively detect new types of attacks. Intrusion detection systems (IDS) fill this role and usually form *the last line of defense* against any kinds of attacks. They are useful not only in detecting successful break-ins, but also in monitoring attempts to breach security.

The thesis presented the use of Anomaly host based IDS to limit the damages caused by malicious VBScript attacks. It is based on two main assumptions: the majority of malicious codes exhibit different behaviors in comparison to normal codes, at the same time the codes within each group share common patterns. The code patterns exhibited in

malicious VBScripts are unique to attack codes and record very low presence in normal VBScripts. The results reached in the thesis have proved both hypotheses.

The proposed anomaly host based IDS uses Principal Component Analysis (PCA), a multivariate statistical technique, that identifies patterns in script codes by highlighting their similarities and differences. Getting relations in this problem that combine a group of VBScript features is computationally expensive and are hard to find. PCA is used to reduce the dimensionality of data in which a large number of variables (243 elements) are interrelated. PCA accomplishes this by computing a smaller set (20 Principal Components) of uncorrelated variables which best represent the original data. Therefore, it can easily detect VBScripts with malicious behavior and allow for normal scripts to bypass.

A working implementation has been developed and tested against real viruses (malicious VBScripts) collected from various sources. Two types of experiments were used to train and test the system: experiment over known VBScript and experiment over unseen VBScript (5 fold cross validation). The testing carried over the developed system showed that the proposed scheme has a good detection rate with low false alarms. The first detection method over seen and unseen data shows that the use of the center of malicious plays an important role in enhancing the overall system accuracy to (94%). However, the limitation of the first detection method is the 26% high rate of false negatives.

The second detection method offers the flexibility of using different threshold points and thus, balancing the false positives and false negatives. The system delivers a good job in detecting VBScript attacks. It has 10% false positive, 29% false negative, and 75% detection rate. Comparing these results with the results of Cunningham [19] who proposes a neural-network system to classify the attacks (developed in C and Unix Shell Scripts), it was found that VBScripts are more difficult to accurately classify than attacks developed in more structured languages (e.g. C). The C detector has 10% false alarms and 98% detection rate. On the other hand, the proposed system yields similar results when compared to Unix Shell detector, it has 80% detection rate.

Finally, the system is robust; it performed well in training and detection. Training the system has a complexity of O (n^2) which is the complexity of matrix conversion. While the system detection which is based on Euclidean distance (a simple distance measure) uses a small feature vectors (20 principal components) in detection. Thus, it doesn't have a bad impact on system performance and computing resources. Both detection method 1 and detection method 2 performed well, they have a complexity of O (n).

6.2 Contributions

The main contribution in this thesis is proposing an extension on Cunningham's work [19] through the detection of source codes of attacks from malicious VBScripts. Script coding such as VBScripts dominate the vast majority of internet script utilization; however, the literature available to present did not sufficiently address the problem of malicious VBScripts as a backdoor of attacking many computers. Thus, using an anomaly host based IDS that limits the damage caused by malicious VBScripts addresses a serious research gap. One of the key benefits of the introduced IDS is that it is not signature-based, therefore it handles a class of malicious codes, and active scripting based attacks rather than specific instances. As a result, it addresses both future and unknown active scripting attacks.

The use of Principal Component Analysis (a Multivariate statistical technique) for extracting benign profile also proved advantageous. First, it does not have any distributional assumption compared to other statistical based intrusion detection methods which usually assume a normal distribution profile. Secondly, it is typical for the data of this problem to be high dimensional. Hence, in the proposed scheme, robust PCA is applied to reduce the dimensionality in order to build a classifier that depends only on the principal components. PCA reduces the dimension of the data without sacrificing its valuable information. Only a few parameters of principal components need to be retained for future detection.

Finally, the proposed classifier is based on a simple distance instead of using Neural Networks (NNs) in classification as followed by Cunningham. The proposed scheme is based on Euclidean distance and thus it is direct and simple. Its advantageous nature

comes from the minimum time it takes to classify. The mean class values are used as class centers; a script is classified as belonging to malicious class, if the distance from benign profile is beyond a certain threshold that can be changed to optimize the operation of the classifier.

6.3 Future work

Future work based on this thesis could be divided into various directions:

1. A key benefit of the proposed system is its low false positive rate of 2% in detection method 1, a misuse IDS system on the other hand usually delivers a low false negative rate. Signature-based systems can identify previously unseen attacks that are abstractly equivalent to known patterns. As they are inherently unable to detect truly novel attacks and suffer from false positive rates. So the primary strength of anomaly detection is its ability to recognize novel attacks. The main drawback though is the difficulty of tracking natural changes; these changes can cause false alarms, while intrusive activities that appear to be normal can cause missed detection. Thus, the introduced IDS can benefit from the advantages of both techniques. Using a combined misuse and anomaly detection will tackle the drawback in each technique. So the introduced IDS could easily detect previous attacks as well as new and unseen ones with low rate of false alarms.

2. Increasing the size of the proposed problem by using more examples of VBScript attacks and dividing the malicious class into several classes such as changing a registry, infection, replication classes. Then, PCA will be applied as a classifier to determine which type of attacks has performed in order to prevent further attacks.

3. Test the use of PCA as a classifier (instead of neural networks) when applied on Cunningham's proposed work [19] and compare its results with the proposed system.

4. The use of PCA as a classifier was chosen based on its simplicity in classification. Other methods of classification that uses different distance metrics, as well as different detection techniques (e.g. neural networks) can be tested to check its impact

on the system's results. There might be a tradeoff between speed of classification and system accuracy.

5. Although VBScript is one of the most widespread scripting languages on the web, other language (e.g., Java) and scripting language (e.g. JavaScript) are still in use and do present significant security holes. The proposed system can be extended by adding different language specific feature extraction.

References

1- D.E. Denning, “An Intrusion Detection Model”, IEEE Transactions on Software Engineering, Vol. SE-13, No 2, Feb. 1987, pp. 222-232.

2- J. McHugh, A. Christie, and J. Allen “Defending Yourself: The Role of Intrusion Detection Systems”, IEEE Software, Sept./Oct. 2000, pp. 42-51.

3- J.P. Anderson, “Computer Security Threat Monitoring and Surveillance”, Tech. Report, James P. Anderson Co, Fort Washington, Pa., 1980.

4- S. Kumar, “Classification and Detection of Computer Intrusion”, Ph.D., Aug. 1995.

5- T.F. Lunt, A. Tarnaru, F. Graham et al., “A Real Time Intrusion Detection Expert System (IDES)”, Final Technical Report, Computer Science Laboratory, SRI International, Menlo Park, California, Feb. 1992.

6- T.F. Lunt, “A Survey of Intrusion Detection techniques”, Computers and Security, 12(4), Jun. 1993, pp.405-418.

7- S. Garfinkel, and G. Spafford, “Practical Unix Security”, O’Reilly and Associates, Sebastopol, California, 1991.

8- E. Amoroso, “Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Trace Back, Traps, and Response”, 1st ed., Intrusion. Net Books, New Jersey, USA, Jan. 1999.

9- H. Debar, M. Dacier, and A. Wespi, “A Revised Taxomomy for Intrusion Detection systems”, Research Report, IBM Research, Zurich Research Laboratory, Switzerland, 1999.

10- C. Nachenberg, “Mobile Code Threats, Fact or Fiction”, International Virus Prevention Conference (IVPC), 1999.

11- J. Allen et al., “State of the Practice of Intrusion Detection Technologies”, Technical Report CMU/SEI_99-TR-) 028, Carnegie Mellon University, Software Engineering Inst., Pittsburgh, 2000.

12- A. Sundaram, “An Introduction to Intrusion Detection”, ACM Crossroads Student Magazine, Apr. 1996.
http://www.acm.org/crossroads/xrds2-4/intrus.html.

13- S.A. Hofmeyer, S. Forrest, and A. Somayaji, "Intrusion Detection Using Sequences of System Calls", Journal of Computer Security, 6, 1998, pp. 151-180.

14- A. Jones, and S. Li, "Temporal Signature for Intrusion Detection", 17th Annual Computer Security Applications Conference (ACSAC'2001), New Orleans, Louisiana, Dec. 2001.

15- W. Lee, S. Stolfo, and P.K. Chan, "Learning Patterns from Unix Process Execution Traces for Intrusion Detection", In Proceedings of AAAI97 Workshop on AI Methods in Fraud and Risk Management, 1997.

16- A. K. Ghosh, J. Wanken, and F. Charron, "Detecting Anomalous and Unknown Intrusions against Programs", In Proceedings of the 1998 Annual Computer Security Applications Conference (ACSAC'98), Dec. 1998.

17- A. K. Ghosh, A. Schwartzbard, and M. Schatz, "Learning Program Behavior Profiles for Intrusion Detection", 1st Workshop on Intrusion Detection & Network Monitoring, Santa Clara, California, USA, Apr. 1999.

18- D. B. Rolsma, "Microsoft Windows Security Patches", Jul. 3, 2001. http://www.sans.org/rr/win/patches.php.

19- R. K. Cunningham, and C. S. Stevenson, "Accurately Detecting Source code of Attacks that increase privilege", 4th International Symposium, Recent Advances in Intrusion Detection (RAID), Oct. 10-12, 2001, pp. 104-116.

20- 'Virus Information", http://www.mcafee.com/anti-virus/default.asp.

21- "Symantec", http://www.symantec.com.

22- M.S. Rich, "Threats to Computer Systems: An Overview", An Introduction to Computer Security: The NIST Handbook, National Institute of Standards and Technology, March 1994.
http://csrc.nist.gov/publications/nistbul/csl94-03.txt

23- G. McGraw, and G. Morrisett, "Attacking Malicious Code: A Report to the Infosec Research Council", IEEE Software, May 1, 2000.

24- R.P. Lippmann, and R.K. Cunningham, "Improving Intrusion Performance using Keywords Selection and Neural networks", Computer Networks, 2000, pp. 597-603.

25- D. Wagner, and P. Soto, “Mimicry Attacks on Host-Based Intrusion Detection Systems”, 9th ACM Conference on Computer and Communication Security (CCS’02), Washington DC, USA, Nov. 18-22 2002.

26- J. Zurada. “Introduction to Artificial Neural Systems”, WEST publishing company, 1992.D.

27- K. Zou, W. Hall, and D. Shapiro. “Smooth non-parametric ROC curves for continuous diagnostic tests.” Statistics in Medicine, 1997.

28- S.L. Scott, “A Bayesian Paradigm for Designing Intrusion Detection Systems”, Computational Statistics and Data Analysis, Jun. 20, 2002.

29- T. Abraham, “Intrusion Detection Using Data Mining Techniques”, Technical Report in Electronics & Surveillance Research Laboratory, 2001.

30- S. Axelsson, “Intrusion Detection Systems: A Survey and Taxonomy”, Technical Report, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, 2000.

31- F. Apap, A. Honig, S. Hershkop, E. Eskin, S. J. Stolfo. ``Detecting Malicious Software by Monitoring Anomalous Windows Registry Accesses." *In Proceedings of the Fifth International Symposium on Recent Advances in Intrusion Detection (RAID-2002)*. Zurich, Switzerland: Oct. 16-18, 2002.

32- M. G. Schultz, E. Eskin, E. Zadok, M. Bhattacharyya, and S. J. Stolfo. ``Malicious Email Filter - A UNIX Mail Filter that Detects Malicious Windows Executables." *In Proceedings of USENIX Annual Technical Conference - FREENIX Track*. Boston, MA: June 2001.

33- M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo. ``Data Mining Methods for Detection of New Malicious Executables" *In Proceedings of IEEE Symposium on Security and Privacy (IEEE S&P-2001)*. Oakland, CA: May 14-17, 2001.

34- J. Kephart, and W. Arnold. “Automatic Extraction of Computer Virus Signatures.” 4th Virus Bulletin International Conference, pp. 178-184, 1994.

35- R. Kohavi. “A Study of Cross-Validation and Boot-Strap for Accuracy Estimation and Model Selection.” IJCAI, 1995.

36- H. Mousa. “A Survey and Analysis of Neural Network Approaches to Intrusion Detection”. Global Information Assurance Certification (GIAC) practical repository, SANS Institute, Nov. 12, 2002.

37- S. White. “Open Problems in Computer Virus Research”, Virus Bulletin Conference, Munich, Germany, Oct. 1998.

38- SysInternals. Regmon for Windows NT /9X Online publication, 2000. http://www.sysinternals.com/ntw2k/source/regmon.shtml

39- J. lapiedra. “The Information Security Process: Prevention, Detection and Response”, Feb.20, 2001. http://www.sans.org/rr/start/process.php.

40- “VBS.ILoveYou (A.K.A. LoveLetter, LoveBug)”, http://getvirushelp.com/iloveyou/

41- Symantec, “Understanding Heuristics: Symantec’s Bloodhound Technology” White Paper Series Volume XXXIV, Nov.30, 1999.

42- McAfee, “Advanced Virus Detection: Scan Engine and DATs, Comprehensive Scanning Technology for today’s threats and tomorrow’s”, Executive White Paper, 2002.

43- P. Verma, ``Virus Protection". Encyclopedia of Information Security, Kluwer (to be published), 2004.

44- VBScript reference, http://www.microsoft.com/resources/documentation/windows/2000/server/scriptguide/en-us/sagsas_overview.mspx

45- V. Anupam, and A. Mayer, “Security of Web Browser Scripting Languages: Vulnerabilities, Attacks, and Remedies”, In Proceedings of the 7th USENIX Security Symposium, San Antonio, Texas, Jan. 26-29, 1998.

46- Y. Nong, S. Emran, Q. Chen, and S. Vilbert, “Multivariate Statistical Analysis of Audit Trails for Host-Based Intrusion Detection”, IEEE Transactions on Computers, Vol. 51, No. 7, Jul. 2002.

47- Y. Nong, and Q. Chen, “An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems.” Quality and Reliability Engineering International, Vol. 17, No. 2, 2001, pp. 105 - 112.

48- Y. Nong, S. Emran, Q. Chen, and K. Noh, “Chi-Square Statistical Profiling for Anomaly Detection”, In Proceedings of the 2000 IEEE Workshop on Information Assurance and Security, United States Military Academy, West Point, NY, Jun. 6-7, 2000.

49- Y. Nong, S. Emran, Q. Chen, and S. Vilbert, “Hotelling’s T^2 Multivariate Profiling for Anomaly Detection”, In Proceedings of the 2000 IEEE Workshop on Information Assurance and Security, United States Military Academy, West Point, NY, Jun. 6-7, 2000.

50- Y. Nong, S. Emran, Q. Chen, X. Li, and M. Xu, “Probabilistic Techniques for Intrusion Detection Based on Computer Audit Data”, IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, Vol. 31, No. 4, Jul. 2001.

51- M. Shyu, S. Chen, K. Sarinnapakon, and L. Chang, “A Novel Anomaly Detection Scheme Based on Principal Component Classifier”, Proceedings of ICDM Foundation and New Direction of Data Mining workshop, 2003, pp. 172-179.

52- Y. Nong, and S. Emran, ”Robustness of Canberra Metric in Computer Intrusion Detection”, Proceedings of the 2000 IEEE Workshop on Information Assurance and Security, United States Military Academy, West Point, NY, Jun. 5-6, 2001.

53- D. Anderson, T. Frivold, and A. Valdes, “Next-Generation Intrusion Detection Expert System (NIDES): A Summary”, Technical Report SRI-CSL-97-07, Menlo Park, Calif.: SRI Int’l, May 1995.

54- H.S. Javitz and A. Valdes, “The SRI Statistical Anomaly Detector”, Proceedings of the 1991 Symposium, Research in Security and Privacy, May 1991.

55- D. Larochelle, and D. Evans,”Statistically Detecting Likely Buffer Overflow Vulnerabilities”, In Proceedings of the 10th USENIX Security Symposium, Aug. 2001.

56- D. Wagner, “Static Analysis and Computer Security: New Techniques for Software Assurance”, University of California, Berkeley, PhD Thesis, 2000.

57- K. I. Diamantaras, S. Y. Kung, “Principal Component Neural Networks : Theory and Applications”, A Wiley-Interscience Publication, John Wiley & Sons, Inc., New York,1996.

58- L. Smith, “A Tutorial on Principal Component Analysis”, Feb. 26, 2002. http://www.cs.rochester.edu/~sprague/446papers/principal_components.pdf

59- D. Cai, J. Theiler, and M. Gokhale, “Detecting a Malicious Executable without Prior Knowledge of its Patterns”, Nonproliferation and International Security, approved for public release distribution unlimited, LA-UR-03.

60- “Ebcvg.com your source for information security”, http://www.ebcvg.com.

61- “VX Heavens”, http://vx.netlux.org/.

62- F. Buchloz, T. Daniels, J. Early et al., “Digging for Worms, Fishing for Answers”, The CERIAS Intrusion Detection Research Group, Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC), 2002.

63- “Principal Component Analysis and Karhunen-loeve transformation”, Pattern Information Pocessing (2001 Autumn Semester), http://kuva.mis.hiroshima-u.ac.jp/~asano/Kougi/02a/PIP/10-25.pdf.

64- H. Anton, “Elementary Linear Algebra 5e”, Publisher John Wiley & Sons Inc., Eighth Edition, New York, 2000.

65- “Slammer: One Year later”, Symantec. Jan. 26, 2004. http://enterprisesecurity.symantec.com/content.cfm?articleid=3261&EID=0

66- “W32.Netsky.C”, Symantec. Feb. 24, 2004. http://securityresponse1.symantec.com/sarc/sarc.nsf/html/w32.netsky.c@mm.html

67- “CERT Advisory 2000-04 Love Letter Worm”, May 4, 2000 http://www.cert.org/advisories/CA-2000-04.html

68- B. Nguyen, “Self Organizing Map for Anomaly Detection”, Advanced Topics in Artificial Intelligence, Spring 2002.

69- “VBScript Language Reference”, SunChili!Soft. ASP Documentation http://17.webmasters.com/caspdoc/html/vbscript_language_reference.htm

70- “MSDN VBScript Language Reference”, http://msdn.microsoft.com/library/default.asp?url=/library/en-vs/script56/html/vbscripttoc.asp

Appendix A

Parser

```
#include "Ftest4.h"
/*******************************************************/
char      Elements[MAXELEMENTS][MAXLINELEN];
int       ElementsRepetition[MAXELEMENTS][MAXFILES];
int       ElementsOccuranceFlag[MAXELEMENTS][MAXFILES];
float     CumulativePercentOccurance[MAXELEMENTS];
float     Correlation[MAXELEMENTS][MAXELEMENTS];
int       LineEvaluate(char *virline,char *elementline);
int       numfiles=0,numelements=0;
/*******************************************************
Program Steps
-------------
1) Open the feature file
2) Create the output file for the analysis
3) Read the elements of the feature
4) Fill the 'ElementsRepetition' matrix
5) Fill the 'ElementsOccuranceFlag' matrix
6) Fill the 'CumulativePercentOccurance' matrix
7) Fill the 'Correlation' matrix
8) Write output file
*******************************************************/
int main(int argc, char* argv[])
{
  FILE                *infile,*outfile,*virfile;
  struct ffblk      filesearchresult;
  int                 filesearchflag;
```

```
int                     i,j,repeatnum;
char                    buffer[MAXLINELEN];
char                    *LineElement;
char                    CompleteFileName[MAXLINELEN];
int                     UptoElementsOccurance[MAXFILES];
float
UptoElementsPercentOccurance[MAXELEMENTS];

LineElement = malloc(MAXLINELEN*sizeof(char));
memset(UptoElementsOccurance,0,MAXFILES*sizeof(int));
memset(UptoElementsPercentOccurance,0,MAXELEMENTS*sizeof(float));

memset(ElementsRepetition,0,MAXELEMENTS*MAXFILES*sizeof(int));
memset(CumulativePercentOccurance,0,MAXELEMENTS*sizeof(float));

if(argc != 4)
{
        printf("\nLooks for the feature in 'in' file then writes \
        the analysis in 'out'");
        printf("\nover the VBS files in directory 'dir'");
  printf("\nFormat : FTEST in out dir\n");
  return 0;
}

// 1 - Open the feature file
if ((infile = fopen(argv[1], "rt")) == NULL)
{
  printf("\nCan't open input file %s. Press any key to exit\n",argv[1]);
  return 1;
}
//////////////////////////////////////////////////
```

```
  // 2 - Create the output file for the analysis
  if ((outfile = fopen(argv[2], "wt")) == NULL)
  {
    printf("\nCannot open output file %s. Press any key to exit\n",argv[2]);
    return 1;
  }
  ////////////////////////////////////////////////
  // 3 - Read the elements of the feature
  while(!feof(infile))
  {
          fgets(Elements[numelements],MAXLINELEN,infile);

          // if the string is too short, do not count
    if(strlen(Elements[numelements])>1)
    {
                  // eliminate last character being '\n'
      if(Elements[numelements][strlen(Elements[numelements])-1] == '\n')
                        Elements[numelements][strlen(Elements[numelements])-1]
= 0;
      StrtoLower(Elements[numelements]);
          numelements++;
          }

          if(numelements == MAXELEMENTS)
    {
          printf("\nMAX number of elements exceeded (%i).\
          Press any key to terminate\n",MAXELEMENTS);
      getch();
          return 1;
    }
  }
```

```
//////////////////////////////////////////////////
// 4 - Fill the 'ElementsRepetition' matrix
sprintf(CompleteFileName,"%s%s",argv[3],"\\*.vbs");
filesearchflag =          findfirst(CompleteFileName,&filesearchresult,0);
while(!filesearchflag)
{

sprintf(CompleteFileName,"%s%s%s",argv[3],"\\",filesearchresult.ff_name);

        if ((virfile = fopen(CompleteFileName,"r")) == NULL)
  {
          printf("\nCannot open file %s. Press any key to\
         exit\n",filesearchresult.ff_name);
    getch();
    fclose(infile);
    fclose(outfile);
    fclose(virfile);
        return 1;
  }

  while(!feof(virfile))
  {
                memset(buffer,0,MAXLINELEN*sizeof(char));
        if(fgets(buffer,MAXLINELEN,virfile) == NULL)
        break;

    StrtoLower(buffer);

    for (i=0;i<numelements;i++)
        ElementsRepetition[i][numfiles]+= LineEvaluate(buffer,Elements[i]);
  }
```

```
  fclose(virfile);

        numfiles++;
        if(numfiles == MAXFILE)
  {
        printf("\nMAX number of files exceeded (%i). Press any key to \
        exit\n",MAXFILES);
    getch();
    fclose(infile);
    fclose(outfile);
    fclose(virfile);
        return 1;
  }

  sprintf(CompleteFileName,"%s%s",argv[3],"\\*.vbs");
        filesearchflag =        findnext(&filesearchresult);
}
////////////////////////////////////////////////
// 5 - Fill the 'ElementsOccuranceFlag' matrix
for(i=0;i<numelements;i++)
        for(j=0;j<numfiles;j++)
        if(ElementsRepetition[i][j])
        ElementsOccuranceFlag[i][j]=1;
////////////////////////////////////////////////
// 6) Fill the 'UptoElementsPercentOccurance' &
//              'CumulativePercentOccurance' matrix
for(j=0;j<numfiles;j++)
        for(i=0;i<numelements;i++)
        if(ElementsRepetition[i][j])
        UptoElementsOccurance[j]++;
```

```
for(j=0;j<numfiles;j++)
        UptoElementsPercentOccurance[UptoElementsOccurance[j]-1]++;

for(i=0;i<numelements;i++)

UptoElementsPercentOccurance[i]=UptoElementsPercentOccurance[i]/numfiles;

for(i=0;i<numelements;i++)
        for(j=numelements-1;j>=i;j--)
    CumulativePercentOccurance[i]+=UptoElementsPercentOccurance[j];

//////////////////////////////////////////////////
// 7 - Fill the 'Correlation' matrix
for(i=0;i<numelements;i++)
        for(j=i;j<numelements;j++)
        Correlation[i][j] = Correlate(ElementsOccuranceFlag[i],
        ElementsOccuranceFlag[j],numfiles);
//////////////////////////////////////////////////
// 8 - Write output file
fprintf(outfile,"\n");
fprintf(outfile,"[Analysis Information]\n");
fprintf(outfile,"======================\n");
fprintf(outfile,"Directory of test files  :  %s\n",argv[3]);
fprintf(outfile,"Number of files tested   :  %i\n",numfiles);
fprintf(outfile,"\n\n");

fprintf(outfile,"Feature file name        :  %s\n",argv[1]);
fprintf(outfile,"Number of Elements       :  %i\n",numelements);
fprintf(outfile,"\n\n\n");
```

```
fprintf(outfile,"[Elements list]\n");
fprintf(outfile,"===============\n");
fprintf(outfile,"     %-60s  Repetition    Occurance\n","Element");
fprintf(outfile,"     %-60s  ----------    ---------\n","-------");
for(i=0;i<numelements;i++)
{
        fprintf(outfile,"%c - '%-60s'     ",'a'+i,StrtoLower(Elements[i]));

  for(repeatnum=0,j=0;j<numfiles;j++)
        repeatnum += ElementsRepetition[i][j];
        fprintf(outfile,"%3i           ",repeatnum);

  for(repeatnum=0,j=0;j<numfiles;j++)
        repeatnum += ElementsOccuranceFlag[i][j];
        fprintf(outfile,"%3i\n",repeatnum);
}
fprintf(outfile,"\n\n\n");


fprintf(outfile,"[Elements Correlation]\n");
fprintf(outfile,"======================\n");

fprintf(outfile,"     ");
for(i=0;i<numelements;i++)
        fprintf(outfile,"  %c     ",'a'+i);
fprintf(outfile,"\n");

fprintf(outfile,"    ");
for(i=0;i<numelements;i++)
        fprintf(outfile,"  ---   ");
fprintf(outfile,"\n");
```

```
  for(i=0;i<numelements;i++)
  {
          fprintf(outfile,"%c : ",'a'+i);
          for(j=0;j<numelements;j++)
    {
                  if(j>=i)
          fprintf(outfile,"%+2.2f  ",Correlation[i][j]);
       else
          fprintf(outfile,"       ");
    }
    fprintf(outfile,"\n");
  }
  fprintf(outfile,"\n\n\n");


  fprintf(outfile,"[Distribution of Elements Occurance]\n");
  fprintf(outfile,"====================================\n");
  fprintf(outfile,"%% of feature hit      %% of code population\n");
  fprintf(outfile,"----------------      --------------------\n");
  for(i=0;i<numelements;i++)
          fprintf(outfile,"     %05.1f%%                %05.1f%%%\n",
          (float)(100*(i+1)/numelements),100*CumulativePercentOccurance[i]);
  ////////////////////////////////////////////////

  fclose(infile);
  fclose(outfile);
  fclose(virfile);

  return 0;
}
```

```
/*******************************************************/
char	*StrtoLower(char	*str)
{
  int	i=0;
  while(str[i]!=0)
		str[i++] = (char)tolower(str[i]);
  return str;
}
/*******************************************************/
float	Correlate(int	*array1,int	*array2,int len)
{
  float		mean1=0.0,mean2=0.0;
  float		numerator=0.0,denom1=0.0,denom2=0.0;
  int		i;

  for(i=0;i<len;i++)
  {
		mean1 += array1[i];
		mean2 += array2[i];
  }

  mean1 = mean1/len;
  mean2 = mean2/len;

  for(i=0;i<len;i++)
		numerator += (array1[i]-mean1)*(array2[i]-mean2);

  for(i=0;i<len;i++)
  {
		denom1 += pow(array1[i],2);
		denom2 += pow(array2[i],2);
```

```
  }

  denom1 -= len*pow(mean1,2);
  denom2 -= len*pow(mean2,2);

  if(denom1 == 0)
        return -9.99;

  if(denom2 == 0)
        return -9.99;

  return(numerator/sqrt(denom1*denom2));
}
/*******************************************************/
char    *TrimSpaces(char    *str)
{
  char  *output = str;
  while((str[strlen(str)-1] == ' ') || (str[strlen(str)-1] == '\t'))
        str[strlen(str)-1] = 0;

  while((output[0] == ' ') || (output[0] == '\t'))
        output ++;

  strcpy(str,output);
  return str;
}
/*******************************************************/
int     LineEvaluate(char *virline,char *elementline)
{
  unsigned      int     i=0;
  int                   j=0,k=0;
```

```
char    subelement[MAXLINELEN] ="";
char    substline[MAXLINELEN] ="";


while(i < strlen(elementline))
{
        if(elementline[i] == '\"')
  {
    j=0;
    i++;
        do
    {
                        subelement[j] = elementline[i];
      i++;
      j++;
    }
    while(elementline[i] != '\"');

    subelement[j] = '\0';
    substline[k] = (strstr(StrtoLower(virline),
        StrtoLower(subelement)) == NULL) ? '0' : '1';
  }
  else
        substline[k] = elementline[i];

  i++;
  k++;
}

return(val(substline));
```

```
}
/*******************************************************/
int         val(char         *expression)
{
  unsigned        int             i;
  int                                     operation;

  char  arg1[MAXLEN]="";
  char  arg2[MAXLEN]="";

  // Removes dummy parentheses , if any .
  RemoveParentheses(expression);

  // If expression = number , return number , else proceed
  for(i=0;i<strlen(expression)-1;i++)
          if( (isdigit(expression[i])==0) && (expression[i]!='.')  )
                  break;

  if( i==(strlen(expression)-1) )
          return(atof(expression));

  if( (operation =          GetLevel(expression,arg1,arg2,1)) == -1 )
          if( (operation = GetLevel(expression,arg1,arg2,2)) == -1 )
          operation = GetLevel(expression,arg1,arg2,3);

  RemoveParentheses(arg1);
  RemoveParentheses(arg2);

  switch(operation)
  {
          case    OR              :               return(val(arg1) || val(arg2));
```

```
            case    AND   :             return(val(arg1) && val(arg2));
  }
}
/*****************************************************/
char      *RemoveParentheses(char    *input)
{
  int                         OpenParentheses=0;
  unsigned       int      i,length_1=strlen(input)-1;

  if( (input[0]!='(') || (input[length_1]!=')') )
            return(input);

  for(i=0;i<length_1;i++)
  {
            switch(input[i])
            {
                  case    '('       :       OpenParentheses++; break;
                  case    ')'       :       OpenParentheses--; break;
            }
            if( OpenParentheses==0 )
                  return(input);
  }
  input[length_1]=0;
  strcpy(input,&input[1]);

  if( (input[0] =='(') && (input[strlen(input)-1] ==')') )
            return(RemoveParentheses(input));
  else
            return(input);
}
/*****************************************************/
```

```
int GetLevel(char *expression,char *arg1,char *arg2,int level)
{
  int   LevelOfParenthese=0;
  char  x[MAXLEN]="";
  char  C1='\0';
  unsigned  int  i;
  unsigned  short int  OP1;

  switch(level)
  {
      case  1  :  C1='+' ; OP1=OR ; break;
      case  2  :  C1='*' ; OP1=AND ; break;
  }

  for(i=0;i<strlen(expression);i++)
  {
      LevelOfParenthese += (expression[i] == '(')? 1:0;
      LevelOfParenthese -= (expression[ i] == ')')? 1:0;

      if((expression[i] == C1)&&(LevelOfParenthese==0))
      {
          strncpy(x,expression,i*sizeof(char));
          strcpy(arg1,x);
          strcpy(x,&expression[i+1]);
          strcpy(arg2,x);
          return(OP1);
      }
  }
  return(-1);
}
/*******************************************************/
```

Appendix B

Mathematical Background

This appendix first introduces mathematical concepts that will be used in Principal Component Analysis (PCA). It covers standard deviation, covariance, eigenvectors and eigenvalues. It is divided into two parts. The first section focuses on Statistics which looks at distribution measurements, or, how the data is spread out. The second section describes Matrix Algebra and looks at eigenvectors and eigenvalues, important properties of matrices that are fundamental to PCA.

1. Statistics

The entire subject of statistics is based around the idea that given a big set of data, and we want to analyze that set in terms of the relationships between the individual points in that data set. We are going to look at a few of the measures we can do on a set of data, and what they tell about the data itself.

1.1 Standard Deviation

To understand standard deviation, we need a data set. Statisticians are usually concerned with taking a sample of a population. To use election polls as an example, the population is all the people in the country, whereas a sample is a subset of the population that the statisticians measure. The great thing about statistics is that by only measuring a sample of the population, we can work out what is most likely to be the measurement if the entire population is used. In this statistics section, we are going to assume that our data sets are samples of some bigger population.

Here's an example set:

X = [1 2 4 6 12 15 25 45 68 67 65 98]

The symbol X is used to refer to this entire set of numbers. If we want to refer to an individual number in this data set, we will use subscripts on the symbol X to indicate a specific number in this data set to indicate a specific number. Eg. X_3 refers to the 3rd number in X, namely the number 4. Note that X1 is the first number in the sequence. Also, the symbol n will be used to refer to the number of elements in the set X. There are number of things that we can calculate about a data set. For example, we can calculate the mean of the sample.

$$\overline{X} = \sum_{i=1}^{n} \frac{X_i}{n}$$

The symbol $\overline{X}$ indicates the mean of the set X. This formula adds up all the numbers and then divide by how many they are.

Unfortunately, the mean doesn't tell a lot about the data except for a sort of middle point. For example, these two data sets have exactly the same mean 10, but are obviously quite different.

[0 8 12 20] and [8 9 11 12]

So what is different about these two sets? It is the spread of the data that is different. The Standard Deviation (SD) of a data set is a measure of how spread out the data is. The definition of SD is the average distance from the mean of the data set to a point. The way to calculate it is to compute the squares of the distance from each data point to the mean of the set, add them all up, divide by n-1 and take the positive square root. As a formula:

$$s = \sqrt{\frac{\sum_{i=1}^{n} (X_i - \overline{X})^2}{(n-1)}}$$

Where S is the usual symbol for standard deviation of a sample.

1.2 Variance:

Variance is another measure of the spread of data in a data set. In fact it is almost identical to standard deviation. The formula is this:

$$s^2 = \frac{\sum_{i=1}^{n} (X_I - \overline{X})^2}{(n-1)}$$

You will notice that this is simply the standard deviation squared; S^2 is the usual symbol for variance of a sample. Both these measurements are measures of the spread of the data.

1.3 Covariance

The previous two measures are purely 1-dimensional. However many data sets have more than one dimension, and the aim of the statistical analysis of these data sets is usually to see if there is any relationship between the dimensions. For example, we might have a data set for both the height of all the students in a class, and the mark they received for that paper. Statistical analysis is performed to see if the height of a student has any effect on their mark. Standard deviation and variance only operate on 1 dimension, so that we could only calculate the standard deviation for each dimension of the data set *independently* of the other dimensions. However, it is useful to have a similar measure to find out how much the dimensions vary from the mean *with respect to each other*. Covariance is such a measure. Covariance is always measured *between* 2 dimensions. If we calculate the covariance between one dimension and *itself*, you get the variance. So if we had a 3 dimensional data set (x, y, z) then we could measure the covariance between the x and y dimensions, the x and z dimensions, and the y and z dimensions. Measuring the covariance between x and x, or y and y, or z and z would give the variance of the x, y, and z dimensions respectively.

The formula for covariance is very similar to the formula for variance. The formula for variance is written like this:

$$\text{var}(X) = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(X_i - \overline{X})}{(n-1)}$$

The formula for covariance

$$COV(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{(n-1)}$$

It is exactly the same except that in the second set of brackets, the X's are replaced by Y's. "For each data item, multiplying the difference between the x value and the mean of x x, by the difference between the y value and the mean of y = Add all these up, and divide by (n-1).

The exact value is not as important as its sign (ie. Positive or negative). If the value is positive, as it is here, then that indicates that both dimensions *increase together*, meaning that, in general, as the number of hours of study increased, so did the final mark. If the value is negative, then as one dimension increases, the other decreases. If we had ended up with a negative covariance here, then that would have said the opposite that as the number of hours of study increased the final mark *decreased*. In the last case, if the covariance is zero, it indicates that the two dimensions are independent of each other.

COV(X,Y) is equal to COV(Y,X), this means that they are exactly the same since the only difference between COV(X,Y) and COV(Y,X) is that $(X_i - \overline{X})(Y_i - \overline{Y})$ is replaced by $(Y_i - \overline{Y})(X_i - \overline{X})$.

1.4 The covariance Matrix

Covariance is always measured between 2 dimensions. If we have a data set with more than 2 dimensions, there is more than one covariance measurement that can be

calculated. For example, in a 3 dimensional data set (dimensions x, y, z), we could calculate COV(X,Y), COV(X,Z), COV(Y,Z).

A useful way to get all the possible covariance values between all the different dimensions is to calculate them all and put them in a matrix. So, the definition of the covariance matrix for a set of data with dimensions is:

$$C^{m\times n} = (c_{i,j}, c_{i,j} = \mathrm{COV}(\mathrm{Dim}_{\mathrm{i}}, \mathrm{Dim}_{\mathrm{J}}))$$

Where $C^{m\times n}$ is a matrix with n rows and n columns, and D_{im_x} is the x^{th} dimension.

The following is an example of the covariance matrix for an imaginary 3 dimensional data set, using the usual dimensions x, y, and z. Then, the covariance matrix has 3 rows and 3 columns, and the values are this:

$$C = \begin{pmatrix} \mathrm{cov}(x,x) & \mathrm{cov}(x,y) & \mathrm{cov}(x,z) \\ \mathrm{cov}(y,x) & \mathrm{cov}(y,y) & \mathrm{cov}(y,z) \\ \mathrm{cov}(x,x) & \mathrm{cov}(z,y) & \mathrm{cov}(z,z) \end{pmatrix}$$

Down the main diagonal, the covariance value is between one of the dimensions and itself. These are the variances for that dimension. The other point is that since $\mathrm{cov}(a,b) = \mathrm{cov}(b,a)$ the matrix is symmetrical about the main diagonal.

2. Matrix Algebra

This section provides a background on the matrix algebra required in PCA. Specifically it focuses on eigenvectors and eigenvalues of a given matrix.

2.1 Eigenvectors

You can multiply two matrices together, provided they are compatible sizes.

Eigenvectors are a special case of this. Consider the two multiplications between a matrix and a vector in the following examples

$$\begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ 3 \end{pmatrix} = \begin{pmatrix} 11 \\ 5 \end{pmatrix}$$

Example of non-eigenvector

$$\begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} \times \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 12 \\ 8 \end{pmatrix} = 4 \times \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

Example of eigenvector

In the first example, the resulting vector is not an integer multiple of the original vector, whereas in the second example, it is exactly 4 times the vector we began with. The vector is a vector in a 2 dimensional space. The vector $\begin{pmatrix} 3 \\ 2 \end{pmatrix}$(from the second example multiplication) represents an arrow pointing from the origin (0,0) to the point (3,2).The other matrix, the square one, can be thought of as a transformation matrix. If we multiply this matrix on the left of a vector, the answer is another vector that is transformed from its original position.

It is the nature of the transformation that the eigenvectors arise from. Imagine a transformation matrix that, when multiplied on the left, reflected vectors in the line y = x. Then we can see that if there were a vector that lay on the line y = x, it's a reflection itself. This vector and all multiples of it would be an eigenvector of that transformation matrix.

The properties of these eigenvectors are:

1. Eigenvectors can only be found for *square* matrices. And, not every square matrix has eigenvectors. And, given an n × n matrix that does have eigenvectors, there are n of them. Given a 3 × 3 matrix, there are 3 eigenvectors.

2. Another property of eigenvectors is that even if we scale the vector by some amount before we multiply it, we still get the same multiple of it as a result, as in the following example:

$$2 \times \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 6 \\ 4 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} \times \begin{pmatrix} 6 \\ 4 \end{pmatrix} = \begin{pmatrix} 24 \\ 16 \end{pmatrix} = 4 \times \begin{pmatrix} 6 \\ 4 \end{pmatrix}$$

Examples of how a scaled eigenvector is still an eigenvector.

This is because if we scale a vector by some amount, all you are doing is making it longer, not changing it's direction.

3. Lastly, all the eigenvectors of a matrix are *perpendicular*, ie. at right angles to each other, no matter how many dimensions we have. Another word for perpendicular, in math talk, is *orthogonal*. This is important because it means that we can express the data in terms of these perpendicular eigenvectors instead of expressing them in terms of the x and y axes.

Another important thing is that when mathematicians find eigenvectors, they like to find the eigenvectors whose length is exactly one. This is because the length of a vector doesn't affect whether it's an eigenvector or not, whereas the direction does. So, in order to keep eigenvectors standard, whenever we find an eigenvector we usually scale it to make it have a length of 1, so that all eigenvectors have the same length. Here's a demonstration in the example above.

$$\begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

is an eigenvector, and the length of that vector is

$$\sqrt{(3^2 + 2^2)} = \sqrt{13}$$

So we divide the original vector by this much to make it have a length of 1.

$$\begin{pmatrix} 3 \\ 2 \end{pmatrix} \div \sqrt{13} = \begin{pmatrix} 3/\sqrt{13} \\ 2/\sqrt{13} \end{pmatrix}$$

2.2 Eigenvalues

Eigenvalues are closely related to eigenvectors, in fact, we saw an eigenvalue like in the following example:

$$\begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} \times \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 12 \\ 8 \end{pmatrix} = 4 \times \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

The amount by which the original vector was scaled after multiplication by the square matrix was the same? In that example, the value was 4. The *eigenvalue* is 4 associated with that eigenvector. No matter what multiple of the eigenvector we took before we multiplied it by the square matrix, we would always get 4 times the scaled vector. So we can see that eigenvectors and eigenvalues always come in pairs.

Appendix C

Principal Components Analysis

This appendix is designed to give an understanding of the process of Principal Component Analysis (PCA). PCA is a useful technique that has found application in fields such as face recognition and image compression, and is a common technique for finding patterns in data of high dimension.

Principal Components Analysis (PCA) is a way of identifying patterns in data, and expressing the data in such a way to highlight their similarities and differences. Since patterns are hard to find in data of high dimension, where the luxury of graphical representation is not available, PCA is a powerful tool for analyzing data. The other main advantage of PCA is that once these patterns were found in the data; we can compress the data by reducing the number of dimensions, without much loss of information. The following section will explain the steps needed to perform PCA on a set of data.

1. Method

Step 1: Subtracting the mean

For PCA to work properly, we have to subtract the mean from each of the data dimensions. The mean subtracted is the average across each dimension. So, all the x values have $\bar{x}$ (the mean of the x values of all the data points) subtracted, and all the y values have $\bar{y}$ subtracted from them. This produces a data set whose mean is zero, See Table 1 and Figure 1.

PCA example data, original data

Data	
x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

Data with the means subtracted

Data Adjust	
x	y
0.69	0.49
-1.31	-1.21
0.39	0.99
0.09	0.29
1.29	1.09
0.49	0.79
0.19	-0.31
-0.81	-0.81
-0.31	-0.31
-0.71	-1.01

Table 1: PCA example data, original data and data with the means subtracted.

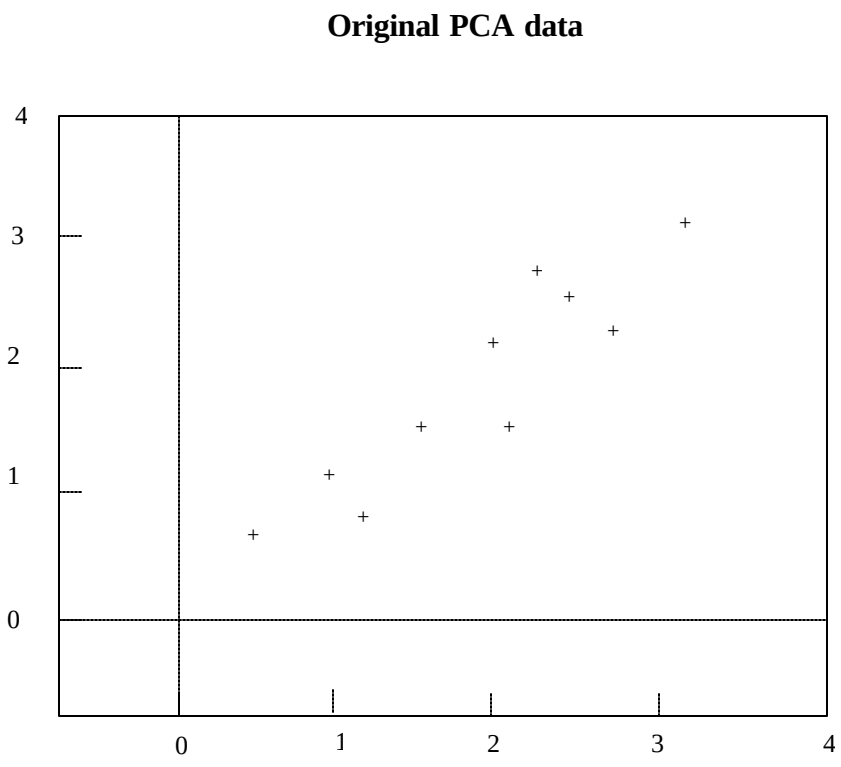


Figure1: Plot of the original data

Step 2: Calculating the covariance matrix

This is done exactly the same way as was discussed earlier. Since the data is 2 dimensional, the covariance matrix will be 2×2.

$$COV = \begin{pmatrix} 0.616555556 & 0.615444444 \\ 0.615444444 & 0.716555556 \end{pmatrix}$$

So, since the non-diagonal elements in this covariance matrix are positive, we should expect that both the X and Y variable increase together.

Step 3: Calculating the eigenvectors and eigenvalues of the covariance matrix

Since the covariance matrix is square, we can calculate the eigenvectors and eigenvalues for this matrix. Here are the eigenvectors and eigenvalues.

$$eigenvalues = \begin{pmatrix} 0.490833989 \\ 1.28402771 \end{pmatrix}$$

$$eigenvectors = \begin{pmatrix} -0.735178656 & -0.677873399 \\ 0.677873399 & -0.735178656 \end{pmatrix}$$

It is important to notice that these eigenvectors are both unit eigenvectors ie. their lengths are both 1. The data has quite a strong pattern. As expected from the covariance matrix, the two variables do indeed increase together. On top of the data, the eigenvectors were plotted. They appear as diagonal dotted lines on the plot. As stated in the eigenvector section, they are perpendicular to each other. They provide information about the patterns in the data. The eigenvectors go through the middle of the points, like drawing a line of best fit. That eigenvector shows how these two data sets are related along that line. The second eigenvector gives the other, less important, pattern in the data, that all the points follow the main line, but are off to the side of the main line by some amount.

So, by this process of taking the eigenvectors of the covariance matrix, we have been able to extract lines that characterize the data. The rest of the steps involve transforming the data so that it is expressed in terms of lines, See Figure 2.

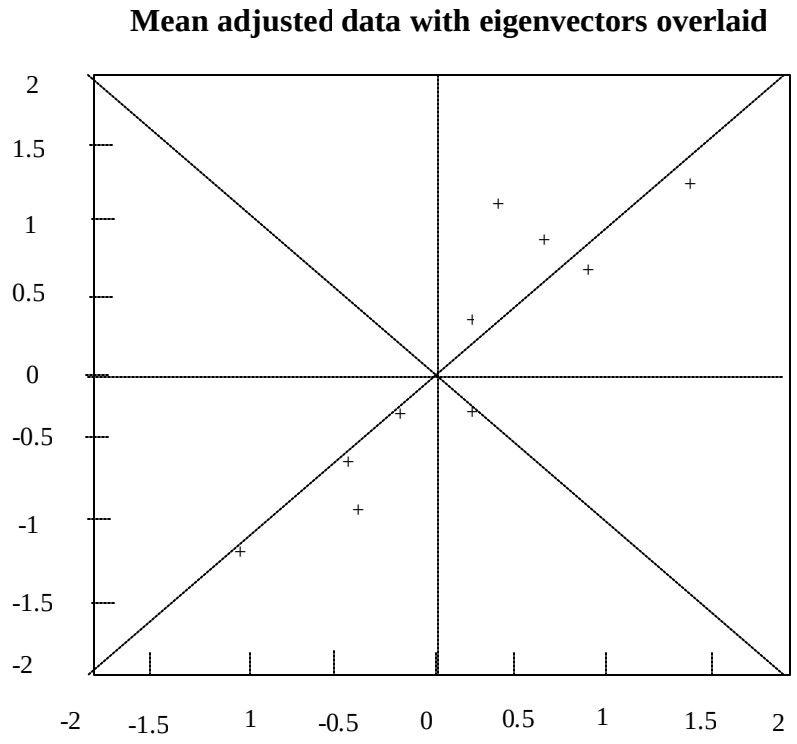


Figure2: A plot of the normalized data (mean subtracted) with the eigenvectors of the covariance matrix overlaid on top.

Step 4: Choosing components and forming a feature vector

Here is where the notion of data compression and reduced dimensionality comes into it. If we look at the eigenvectors and eigenvalues from the previous section, we will notice that the eigenvalues are quite different values. In fact, it turns out that the eigenvector with the *highest* eigenvalue is the *principle component* of the data set.

In our example, the eigenvector with the largest eigenvalue was the one that pointed down the middle of the data. It is the most significant relationship between the data dimensions.

In general, once eigenvectors are found from the covariance matrix, the next step is to order them by eigenvalue, highest to lowest. This gives you the components in order of significance. Now, if you like, you can decide to *ignore* the components of lesser significance. You do lose some information, but if the eigenvalues are small,

you don't lose much. If you leave out some components, the final data set will have less dimensions than the original, see Table 2 and Figure 3.

To be precise, if you originally have n dimensions in your data, and so you calculate n eigenvectors and eigenvalues, and then you choose only the first P eigenvectors, then the final data set has only P dimensions. What needs to be done now is you need to form a feature vector for a matrix of vectors. This is constructed by taking the eigenvectors that you want to keep from the list of eigenvectors, and forming a matrix with these eigenvectors in the columns.

$$\text{FeatureVector} = (\text{eig}_1\ \text{eig}_2\ \text{eig}_3 \ldots\ldots\ldots\ldots..\text{eig}_n)$$

Given our example set of data, and the fact that we have 2 eigenvectors, we have two choices. We can either form a feature vector with both of the eigenvectors.

$$\begin{pmatrix} -0.677873399 & -0.735178656 \\ -0.735178656 & 0.677873399 \end{pmatrix}$$

or, we can choose to leave out the smaller, less significant component and only have a single column.

$$\begin{pmatrix} -0.677873399 \\ -0.735178656 \end{pmatrix}$$

Transformed Data	
x	y
-0.827970186	-0.175115307
1.77758033	0.142857227
-0.992197494	0.384374989
-0.274210416	-0.209498461
-0.167580142	0.175282444
0.991094375	-0.349824698
1.14457276	-0.349824698
0.438046137	0.177646297
1.22382056	-0.162675287

Table2: Data transformed with 2 eigenvectors

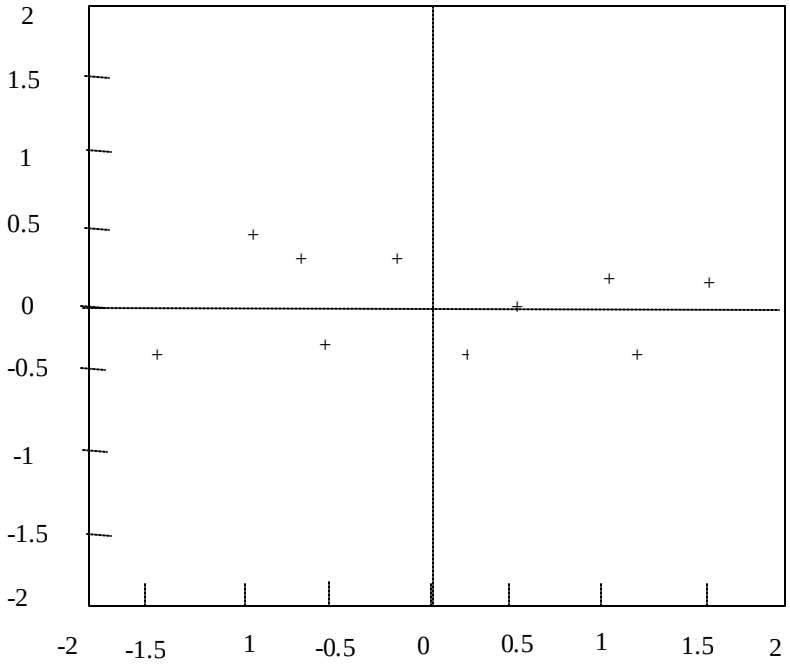


Figure 3:The table of data by applying the PCA analysis using both eigenvectors, and a plot of the new data points.

Appendix D

Malicious VBScript
VBS1.vbs
VBS.TripleSix.vbs
I-Worm.Fiume.vbs
IIS-Worm.IISWorm.vbs
Script.ASX.Conp.vbs
Wordsworth.vbs
IRC-Worm.Wordsworth.vbs
VBS.Monopoly.VBS
MONOPOLY.VBS
VBS.Nomek.vbs
JS.Judge.vbs
VBS.Monopoly.b.VBS
VBS.Monopoly.c.vbs
Monopoly.c.vbs
LovingYou.vbs
VBS.Devolve.vbs
virus.vbs
VBS.Hard.vbs
Hard.vbs
Winvader.vbs
VBS.Charm.vbs
Zulud.vbs
VBS.Zulu.d.vbs
Zulu.d.vbs
VBS.Zuluc.vbs
VBS.Zulu.c.vbs
Alcau.vbs
Zulu.c.vbs
virus8.vbs
Euro2002.vbs
VBS.Trojan.Toren.vbs
Script.BRB.vbs
I-Worm.LoveLetter.VBS
I-A93F~1.VBS
Redlof.vbs
Trip.vbs
VBS.Zulub.vbs
VBS.Zulu.b.VBS
I-Worm.WinXP.vbs
I-Worm.Croatia.vbs
VBS.Casel.vbs

VBS3.vbs
Zulu.b.vbs
I-Worm.Avalon.vbs
VBS.Missy.VBS
missy.vbs
IRC-Worm.Missy.VBS
Britney.vbs
VBS.Trojan.Noob.vbs
VBS.Tunef.vbs
VBS.LoveLetter.A.vbs
VBS.Charlene.vbs
deepblue001.vbs
VBS.Freelink.vbs
I-Worm.Lee.VBS
IIS-Worm.CodeGreen.vbs
Zulu.vbs
VBS.Zulu.VBS
VBS.Phram.b.VBS
VBS.Sant.vbs
Sant.vbs
VBS.Phram.vbs
Phram.vbs
VBS.Chango.vbs
VBS.Saje.vbs
VBS.Mbop.d.vbs
Mbop!-Vbs.vbs
VBS.Enc.vbs
ZELDA.VBS
VBS.Tune.i.vbs
VBS.Tuneb.vbs
Tune.vbs
Kristen.vbs
Mbop!.vbs
VBS.Dream.b.vbs
Cheese.vbs
tripple.vbs
VBS.Rabfu.vbs
VBS.Stuckb.vbs
vbs.dropper.vbs
VBS.Fela.b.vbs
Fela.b.vbs
anna.vbs
VBS.Navigator.vbs
PORN_Madonna.JPEG.vbs
VBS.Trojan.Vanina.b.vbs
vbs.solved.vbs
I-Worm.Noon.vbs
JS.JDV.b.vbs

IRC-Worm.generic.doc.VBS
VBS.Sheep.VBS
VBS.Cute.vbs
IRC-Worm.generic.vbs.vbs
Welcomb.vbs
IRC-Worm.generic.vbs
VBS.Reality.VBS
VBS.Jesus.vbs
Crystal.c.vbs
I-691D~1.VBS
JS1.vbs
I-Worm.LoveLetter.bk.VBS
VBS.Unreal.vbs
VBS.Bogus.vbs
Bogus.vbs
VBS.Sunfl.vbs
VBS.NMVT.vbs
VBS.Jadra.vbs
Like_A_Virgin.MP3.vbs
JS.Fmtdrv.b.vbs
VBS.LaMEr0nE.vbs
VBS.Hustle.vbs
VBS.ChiQui.vbs
Hustle.vbs
VBS.Harana.vbs
VBS.IntendedDrambui.vbs
VBS.Intended.Drambui.vbs
virus2.vbs
kileer.VBS
VBS.Yozisa.vbs
VBS.Yozis.vbs
Inzane.vbs
VBS.Crystal.b.VBS
Crystal.a1.vbs
VBS.Crystal.c.vbs
Crazzzy.c.vbs
Sheep.vbs
VBS.Bound.vbs
MSBound.vbs
JS.Charlene.vbs
VBS.Crystal.vbs
Crystal.vbs
Crysta.vbs
VBS.Dropper.Apop.vbs
Beware.vbs
VBS.Energ.vbs
IIS-Worm.CodeRed.a.vbs
VBS.Stress.vbs

Stress.vbs
Stress Out.vbs
VBScript.777.vbs
VBS.777.vbs
IIS-Worm.CodeRed.c.vbs
Seven.vbs
VBS.Vintageb.vbs
sargo.vbs
Jackal.VBS
Osterhase.vbs
MarkerC.vbs
VBS.Trojan.Zirkov.vbs
Trojan.Zirko.vbs
pet_tick.vbs
VBS.Hatred.b.vbs
Hatred.b.vbs
VBS.GaScript.vbs
IRC-Worm.Freenet.VBS
VBS.LostGame.vbs
JS.Hatred.vbs
VBS.Mesut.vbs
VBS.Both.vbs
VBS.Dropper.17th.vbs
Crazzzy.b.vbs
Renegy.vbs
Fasan.vbs
VBS.Yozisb.vbs
VBS.Yozis.b.vbs
thief.vbs
Intended.Yozis.b. vbs
VBS.Phybre.vbs
I-Worm.Lee.o.vbs
VBS.Simona.vbs
Simona.vbs
VBS.Voodoo1b.vbs
VBS.Voodoo.b.vbs
VBS.Vodoo.b.VBS
VBS.Godog.vbs
Falling.vbs
vacuna.vbs
VBS.Netlog.vbs
JS.Internal.f.vbs
VBS.Trojan.Cybers.vbs
NoWarn.vbs
HTML.NoWarn.vbs
VBS.Trojan.Raptor.vbs
Raptor V.vbs
JS.Germinal.vbs

VBS.Voodoo.vbs
VBS.Vodoo.VBS
VBS.Exposed.vbs
VBS.VbsDoc.vbs
HTMLStertor.vbs
VBS.Petik.vbs
VBS.Trojan.Lava.vbs
dome.vbs
yourworm.vbs
I-Worm.SSIWG.vbs
VBS.Spaced.vbs
VBS.SSIWG.vbs
I-Worm.Sint.src.VBS
VBS.Sflus.vbs
Homepage.vbs
JS.Lame.vbs
Happy.vbs
JS.Evil.vbs
VBS.Coldape.vbs
VBS.AVM.vbs
VBS.Pocus.vbs
VBS.Kremp.VBS
VBS.Internal.vbs
Rato.vbs
fuckosama.vbs
VBS.Lanus.vbs
NEOKILLER.vbs
VBS.Hatred. vbs
Hatred.vbs
VBS.Marata.vbs
HTML.Internal.vbs
AnnaK.vbs
VBS.Weim.vbs
VBS.RTFinfo.vbs
VBS.Entice.vbs
Entice.vbs
WBS.Simple.VBS
Crazzzy.vbs
VBS.Jertva.vbs
JS.Optiz.vbs
VBS.Gum.vbs
VBS.Infi.vbs
madafaka.vbs
Infi.vbs
VBS.Mb. c.VBS
VBS.Madafaka.vbs
Infi.b.vbs
Xxx.vbs

VBS.Xxx.VBS
VBS.Trojan.Alal.vbs
Eraser.vbs
JS.Funtime.vbs
Simple.vBS
VBS.MailTest.vbs
Yovp.vbs
VBS.Internalc.vbs
VBS.Internal.c.vbs
Internal.vbs
JS.vbs
VBS.Eva.b.vbs
Script.Inf.Demo. vbs
VBS.Nobyl.vbs
VBS.Eva.c.vbs
IIS-Worm.BlueCode.vbs
Script.Inf.Zox.c.vbs
VBS.Lucky2.vbs
VBS.Eva.vbs
Eva.vbs
Script.Inf.Zox.b.vbs
VBS.Rabbit.a.VBS
VBS.Rabbit.c.vbs
VBS.Corrupt.vbs
rabbitC.vbs
Baby.vbs
VBS.Trojan.Over.vbs
Script.HE.Flys.b.vbs
VBS.Rabbit.b.vbs
rabbitB.vbs
VBS.Rabbit.vbs
rabbitA.vbs
VBS.Brat.vbs
VBS.Trojan.Destroyer.Trojan.vbs
Fuck.vbs
JS.DriveFormat.b.vbs
VBS.Readme.vbs
Chantal.vbs
VBS.Pie.b.vbs
Script.HE.Flys.vbs
VBS.Trojan.Debor.vbs
Bored.vbs
Small.vbs
VBS.Birgit.vbs
VBS.First.e.vbs
VBS.First.b.VBS
Stuck.vbs
JS.Fortnight.a.vbs

melissa.vbs
Script.Inf.Zox.vbs

Normal VBScripts
zoom.vbs
WSH__I386_SHOWVAR.VBS
WSH__I386_SHORTCUT.VBS
WSH__I386_REGISTRY.VBS
WSH__I386_NETWORK.VBS
WSH__I386_EXCEL.VBS
WSH__I386_DELUSERS.VBS
WSH__I386_CHART.VBS
WSH__I386_ADDUSERS.VBS
Write to a Custom Event Log Using EventCreate.vbs
Write Events to the Local Event Log.vbs
Write Events to a Remote Event Log.vbs
Write Data to a Text File.vbs
Win2kReleaseIPRenewIP.vbs
viewpagefilesettings.vbs
View System Restore Configuration Values.vbs
View All Existing Restore Points.vbs
Vier.vbs
version.vbs
Verify that a Folder Exists.vbs
Verify that a File Exists.vbs
Verify Signatures for All Scripts in a Folder.vbs
Verify ChkDsk Volume Status.vbs
VDirHits.vbs
Vbstest21.vbs
Vbstest2.vbs
vbscript.vbs
Using WMI to Enumerate Shortcuts on a Computer.vbs
uservalid.vbs
UserPrintQueues.vbs
UserPrint.vbs
UserLogon.vbs
UserGroups.vbs
UserFileAccess.vbs
UserChangePassWord.vbs
Use WMI to Enumerate Environment Variables.vbs
Use Wildcards in File Queries.vbs
Use Wildcards in a Folder Query.vbs

Use the Browse for Folder Dialog Box.vbs
Use Recursion to Enumerate Subfolders.vbs
Use a Text File as a Command Line Argument.vbs
Use a Predefined Indexing Service Query.vbs
Use a Free Text Search with the Indexing Service.vbs
Upgrade Software.vbs
update_ifl501.vbs
update_ifl50.vbs
update_icl501.vbs
update_icl50.vbs
update_data_mdb1.vbs
update_data_mdb.vbs
Update Records in a Recordset.vbs
Update Printer Locations.vbs
UnPauseWebServers1.vbs
UnPauseWebServers.vbs
UnPauseServers.vbs
UnPauseFTPServers.vbs
uninstall_admin1.vbs
uninstall_admin.vbs
UNIFLEX.VBS
Uncompress Folders.vbs
U111.VBS
Transfer1.vbs
Transfer.vbs
Transfer Print Jobs to a Different Print Queue.vbs
Track Script Progress Using Internet Explorer.vbs
Track Script Progress in a Command Window.vbs
Top20URIs.vbs
TEXTMASK.vbs
TestOcx2.vbs
TestOcx1.vbs
TestOcx.vbs
testflds1.vbs
testflds.vbs
test_script.vbs
test.vbs
Terminate a Process.vbs
System Event Log Properties.vbs
synciwam.vbs
switchToSourceSafe.vbs
Suppressing Windows Activation Notices.vbs
Suppress Multiple Event Notifications.vbs
submenus98.vbs
StopWebServers.vbs
stopweb.vbs
stopsrv1.vbs
stopsrv.vbs

stopservice.vbs
StopServers.vbs
StopFTPServers.vbs
stopftp.vbs
Stop the Indexing Service.vbs
Stock.vbs
StaticFilter.vbs
StartWebServers.vbs
startweb.vbs
startsrv1.vbs
startsrv.vbs
startservice.vbs
StartServers.vbs
StartFTPServers.vbs
startftp.vbs
Start the Indexing Service.vbs
standardPro8.vbs
standardPro6.vbs
standardPro5.vbs
standardPro1.vbs
standardPro.vbs
standard.vbs
sssadmin7.vbs
sssadmin6.vbs
SSSADMIN3.VBS
SSSADMIN2.VBS
SSSADMIN1.VBS
sssadmin.vbs
SrcSafe.vbs
SPLINELOGO.vbs
SPINIE.vbs
sp500w98pcmciab001.vbs
SMap_Topframe.vbs
SMap_Menu.vbs
SMap_MainCode.vbs
SMap_LinkCode.vbs
Skin_18.vbs
Skin_17.vbs
Skin_16.vbs
Skin_13.vbs
Skin_12.vbs
Skin_11.vbs
Skin_1.vbs
SIM100_Shared1.vbs
SIM100_Shared.vbs
sign.vbs
Sign All Scripts in a Folder.vbs
Sign a Script Programmatically.vbs

shutdown.vbs
Shut Down the Local Computer.vbs
SHOWVAR3.VBS
SHOWVAR2.VBS
SHOWVAR1.VBS
SHOWVAR.VBS
show-ids1.vbs
show-ids.vbs
ShowAllRoutingRules.vbs
ShowAllProtocolRules.vbs
SHORTCUT3.VBS
SHORTCUT2.VBS
SHORTCUT1.VBS
SHORTCUT.VBS
SG-PRESERVE.vbs
SG-DRAWINGSURFACE.vbs
SetWebPubRuleHostHeader.vbs
SetUpstreamRouting.vbs
setup1.vbs
setup.vbs
SetTempPath.vbs
SetEmailAlerts.vbs
SetDefaultApplicationSettings.vbs
SetCache.vbs
set_rwin1.vbs
set_rwin.vbs
Set the Default Printer.vbs
Set the Default Printer Based on Queue Length.vbs
Set System Restore Configuration Values.vbs
servicestopped.vbs
SendText.vbs
SendMail1.vbs
SendMail.vbs
Send Email without Installing the SMTP Service.vbs
Send Email from a Script.vbs
Security Log Properties.vbs
SearchTree.vbs
search_for_emails.vbs
search_for_addresses.vbs
Search the Indexing Service.vbs
Search for Specific Printers.vbs
Search for Published Folders.vbs
Search for Computer Accounts.vbs
SCROLLSWITCH.vbs
Script99.vbs
Script5.vbs
Schedule AutoChk.vbs
Schedule a Task.vbs

RUNNERS.vbs
Run ChkDsk.vbs
Returning Active Directory System Information.vbs
Return a Disk Drive Collection.vbs
Retrieving the Default WMI Namespace.vbs
Retrieve USB Hub Information.vbs
Retrieve USB Controller Information.vbs
Retrieve Time Zone Information for a Computer.vbs
Retrieve the Internet Explorer Connection Summary.vbs
Retrieve the Display Controller Configuration.vbs
Retrieve System Information.vbs
Retrieve Summary Information for a File.vbs
Retrieve Specific Events from an Event Log.vbs
Retrieve SMBIOS Information.vbs
Retrieve Shadow Copy Settings.vbs
Retrieve SCSI Controller Information.vbs
Retrieve Print Queue Statistics.vbs
Retrieve Portable Battery Information.vbs
Retrieve Port Resource Information.vbs
Retrieve PCMCIA Controller Information.vbs
Retrieve Operating System Properties.vbs
Retrieve Motherboard Device Information.vbs
Retrieve Modem Information.vbs
Retrieve Memory Array Information.vbs
Retrieve IP4 Route Table Information.vbs
Retrieve Internet Explorer File Version Information.vbs
Retrieve Internet Explorer Connection Settings.vbs
Retrieve Internet Explorer COM Object Settings.vbs
Retrieve Internet Explorer Cache Settings.vbs
Retrieve Information About Mapped Network Drives.vbs
Retrieve Information About Binary Files Used by an Application.vbs
Retrieve IDE Controller Information.vbs
Retrieve Folder Properties.vbs
Retrieve Floppy Controller Information.vbs
Retrieve File Version Information.vbs
Retrieve File Properties.vbs
Retrieve Extended File Properties.vbs
Retrieve Events For One Day from an Event Log.vbs
Retrieve Detailed Summary Information for a File.vbs
Retrieve Current Display Configuration.vbs
Retrieve Computer Fan Information.vbs
Retrieve Command Line Arguments from an Active Directory Co.vbs
Retrieve Command Line Arguments from a Text File.vbs
retrieve CACHE MEMORY information.vbs
retrieve BIOS information.vbs
retrieve battery information.vbs
Retrieve All Events from an Event Log.vbs
resumeservice.vbs

Resume the Indexing Service.vbs
Resume Print Jobs.vbs
Resume All Paused Printers.vbs
Resume a Paused Printer.vbs
RestorePointXPCleaner1.vbs
RestorePointXPCleaner.vbs
restart1.vbs
restart.vbs
restart a computer.vbs
resource2_h.vbs
resource1_h.vbs
resource_h.vbs
reseta computer account password.vbs
Report Print Queue Statistics.vbs
Rename Folders.vbs
Rename Files.vbs
Rename a Printer.vbs
Rename a Printer Published in Active Directory.vbs
Rename a Folder.vbs
Rename a File.vbs
rename a computer account1.vbs
rename a computer account.vbs
remove.vbs
remove sw.vbs
Remove an Indexing Service Scope.vbs
Remove an Indexing Service Catalog.vbs
Remove a Printer Connection.vbs
remoteshutdown.vbs
remotereboot.vbs
RemoteFFT.VBS
REGISTRY3.VBS
REGISTRY2.VBS
REGISTRY1.VBS
REGISTRY.VBS
RegFilters.vbs
Receive Notification When a Printer Stops.vbs
reboot.vbs
read_cd_root.vbs
Read String and DWORD Values.vbs
Read Data from a Text File.vbs
Read an Expanded String Value.vbs
Read a Text File into an Array.vbs
Read a Text File from the Bottom Up.vbs
Read a Text File Character by Character.vbs
Read a MultiString Value.vbs
Read a Binary Registry Value.vbs
ras1.vbs
ras.vbs

Query the Indexing Service for File Names.vbs
Query the Event Log for Stop Events.vbs
Query a Specific Event Log.vbs
Purge a Print Queue.vbs
Publish a Shared Folder.vbs
Programmatically Verify a Script Signature.vbs
process.vbs
prncnfg.vbs
printjobs.vbs
printerstatus.vbs
Prevent AutoChk From Running.vbs
Prevent a Process from Running.vbs
ppt_name.vbs
Post.vbs
Pierwias1.vbs
Pierwias.vbs
Photoshop_shortcut_xp2k.vbs
Perform Actions on Files.vbs
perform a cross-domain.vbs
pcconfig.vbs
PauseWebServers.vbs
pauseweb.vbs
pausesrv.vbs
pauseservice.vbs
PauseServers.vbs
PauseFTPServers.vbs
pauseftp.vbs
Pause the Indexing Service.vbs
Pause Printers with Empty Print Queues.vbs
Pause Print Jobs.vbs
Pause a Printer.vbs
patch.vbs
Parse a Path Name.vbs
Parse a Fixed Width Column Log.vbs
Parse a Comma Separated Values Log.vbs
pagefileconfig.vbs
pagefile.vbs
osinfo.vbs
ntevent.vbs
NSETUPWS.VBS
Notify Users of a Purged Print Queue.vbs
Notepad_shortcut_xp2k.vbs
NewWindow.vbs
NETWORK3.VBS
NETWORK2.VBS
NETWORK1.VBS
NETWORK.VBS
Move Folders Using WMI.vbs

Move Folders Using the Shell Object.vbs
Move Files.vbs
Move a Set of Files.vbs
Move a Folder.vbs
Move a File.vbs
move a computer account.vbs
mouset.vbs
MOUSERS2.VBS
MOUSERS1.VBS
MOUSERS.VBS
Mount a Volume.vbs
Monitor Volume Change Events.vbs
Monitor Threads.vbs
Monitor the Print Service.vbs
Monitor Registry Subtree Events.vbs
Monitor Registry Subkey Events.vbs
Monitor Registry Entry Level Events.vbs
Monitor Processor Use.vbs
Monitor Processor Use by Process.vbs
Monitor Process Performance.vbs
Monitor Process Deletion. vbs
Monitor Process Creation.vbs
Monitor Process Availability.vbs
Monitor Printers with a Temporary Event Subscription.vbs
Monitor Printer Status.vbs
Monitor Print Queues.vbs
Monitor Print Queue Times.vbs
Monitor Print Job Status.vbs
Monitor Physical Disk Drive Performance.vbs
Monitor Page File Use.vbs
Monitor NTFS File Cache Performance.vbs
Monitor Logical Disk Drive Performance.vbs
Monitor FRS Replication.vbs
Monitor File Modification.vbs
Monitor File Deletion.vbs
Monitor File Creation.vbs
Monitor Event Logs.vbs
Monitor Domain Controller Performance.vbs
Monitor Disk Drive Free Space.vbs
Monitor Disk Bytes Per Second.vbs
Monitor Computer Uptime.vbs
Monitor Computer Health.vbs
Monitor Computer Availability.vbs
Monitor Changes in Service Status.vbs
Monitor Changes in Computer Power Status.vbs
Monitor Available Memory.vbs
Monitor Available Disk Space.vbs
Monitor Active Directory Replication.vbs

Monitor Active Directory Database Performance.vbs
MOGRP2.VBS
MOGRP1.VBS
MOGRP.VBS
Modify the AutoChk Timeout Value.vbs
modify recovery configuration options.vbs
Modify File System Properties.vbs
Modify File Attributes.vbs
modify a shadow copy storage area.vbs
Modify a Network Share.vbs
Modify a Disk Quota Entry.vbs
mod_user.vbs
mkwebsrv.vbs
mkwebdir.vbs
mkw3site.vbs
mkftpsite.vbs
mkftpdir.vbs
MimeMaps.vbs
Migan1.vbs
Migan.vbs
MetaBackRest1.vbs
metabackrest.vbs
MetaBackEnum.vbs
MetaBackDel.vbs
MetaBack1.vbs
metaback.vbs
memory.vbs
MEGA.vbs
mc1.vbs
mc.vbs
Mask Passwords Using Internet Explorer.vbs
Mask Command Line Passwords.vbs
Map All Network Shares to Local Folders.vbs
Map a Network Share to a Local Folder.vbs
machinename.vbs
lznac1.vbs
lznac.vbs
LVDemo.VBS
LoveCleaner1.vbs
LoveCleaner.vbs
Lotto.vbs
logfix3.vbs
logfix2.vbs
logfix1.vbs
logfix.vbs
logenum.vbs
listservices.vbs
ListServers.vbs

listnetworkprotocols.vbs
Listing7_OUAnlegen2.vbs
Listing7_OUAnlegen1.vbs
Listing7_OUAnlegen.vbs
Listing6_Benutzeranlegen1.vbs
Listing6_Benutzeranlegen.vbs
Listing5_Query1.vbs
Listing5_Query.vbs
Listing4_Benutzerliste_Standardcontainer1.vbs
Listing4_Benutzerliste_Standardcontainer.vbs
Listing3_WellKnownObjects.vbs
Listing2_GUID.vbs
Listing1_Partionen.vbs
listeventsbycode.vbs
listeventlogfiles.vbs
ListDeniedMethods.vbs
List Registry Files.vbs
LINKS.VBS
libhdr.vbs
launcher1.vbs
launcher.vbs
LangVer.vbs
LabVIEW_FFT.VBS
Kontrol1.vbs
Kontrol.vbs
join computer domain.vbs
iss.vbs
ISAInfo.vbs
InstCtrs.vbs
install_admin.vbs
install sw on the local computer.vbs
install sw on a remote computer.vbs
Install Printer Ports.vbs
Install Printer Drivers.vbs
Install Multiple Printers for One Print Device.vbs
install computer hw.vbs
Install Active Directory Database Performance Counters.vbs
Install a Printer.vbs
Install a Printer Driver not Found in Drivers Cab.vbs
ims5.vbs
ims4.vbs
ims3.vbs
ims2.vbs
ims1.vbs
ims.vbs
image.vbs
iisftp.vbs
iiscfg1.vbs

iiscfg.vbs
IHelpUtil.vbs
identifyprocessor type.vbs
Identifying the Owner of a Group.vbs
Identifying Attributes that are Both Indexed and in the Glo.vbs
identify windows product activation status.vbs
identify THE LATEST INSTALLED SERVICE PACK.vbs
Identify the Indexing Service State.vbs
Identify the File System Type.vbs
Identify Shell Object Verbs.vbs
identify os.vbs
identify FSMO ROLES.vbs
Identify Drive Types.vbs
identify domain controller.vbs
identify a global catalog server.vbs
identify a computer roles.vbs
identify a computer roles. using services.vbs
identify a computer chassis.vbs
HOST_BAN.VBS
Hex1.vbs
Hex.vbs
hello.vbs
hehe.vbs
hdr1.vbs
hdr.vbs
Hangman.vbs
HackerScan.vbs
GTHRLOG.VBS
GLOBE.vbs
global.vbs
ghdr1.vbs
ghdr.vbs
GetWebPage.vbs
getnetworkadapterconfig.vbs
getbootconfig.vbs
getappboost.vbs
get the results of the last system remoter.vbs
Generate a File Name.vbs
general.vbs
ftp1.vbs
ftp. vbs
FpcCfg.vbs
FORMS.VBS
Format a Volume.vbs
fix1.vbs
fix.vbs
findweb.vbs
FindScheduledContentDownload.vbs

Finding Folders by Date.vbs
Find a Record in a Recordset.vbs
FILTER.vbs
filefind.vbs
FetchUrl.vbs
fcountry1.vbs
fcountry.vbs
Favorites.vbs
fastplay.vbs
Fade_text.vbs
ExtendSchema.vbs
EXCEL5.VBS
EXCEL3.VBS
EXCEL2.VBS
EXCEL1.VBS
EXCEL.VBS
evtquery.vbs
eventquery.vbs
Event Log Properties.vbs
eshmts.vbs
ErrorHandler.vbs
ErrorCodes.vbs
Enumerating WMI Providers.vbs
Enumerating WMI Namespaces.vbs
Enumerating User Accounts on the Local Computer.vbs
Enumerating the Dial-In Property Configuration for a User A.vbs
Enumerating Qualifiers for a WMI Class.vbs
Enumerating Properties, Methods, and Qualifiers for a WMI Class.vbs
Enumerating Properties for a WMI Class.vbs
Enumerating Local Groups and Their Members.vbs
Enumerating Group Members.vbs
Enumerating Floppy Disk Drive Properties.vbs
Enumerating Dynamic Classes in WMI.vbs
Enumerating Disk Space by User on the Local Computer.vbs
Enumerating Disk Quotas on the Local Computer.vbs
Enumerating Disk Quota Settings on the Local Computer.vbs
Enumerating COM+ Partition Sets.vbs
Enumerating CD-ROM Properties.vbs
Enumerating Auxiliary Classes.vbs
Enumerating Audit Permissions for a User Account.vbs
Enumerating All WMI Namespaces.vbs
Enumerating All the User Accounts in an NT 4.0 Domain.vbs
Enumerating All the Attributes of an Active Directory Class.vbs
Enumerating Abstract Classes in WMI.vbs
EnumerateUser.vbs
enumerateserial port config properties.vbs
enumerateserial port properties.vbs
enumerateall domain controllers.vbs

enumerate WMI settings.vbs
Enumerate Volume Properties.vbs
Enumerate Volume Mount Points.vbs
enumerate video controller.vbs
Enumerate Trust Relationships.vbs
enumerate thephysical memory.vbs
Enumerate the Subfolders of a Folder.vbs
Enumerate the Names of Objects in the Configuration Container.vbs
enumerate the codec files on a computer.vbs
enumerate the boot config properties.vbs
enumerate system slot.vbs
Enumerate Subkeys.vbs
Enumerate Subfolders in a Folder.vbs
enumerate start menu.vbs
enumerate start menu prog groups.vbs
Enumerate Special Folders.vbs
enumerate sound card.vbs
enumerate shadow copy providers.vbs
Enumerate Scheduled Tasks.vbs
enumerate RSOP sessions .vbs
enumerate RSOP appls .vbs
enumerate Resultant user privilege right.vbs
enumerate Resultant system services.vbs
enumerate Resultant set of policyGPOs.vbs
enumerate Resultant set of policy RSop.vbs
enumerate Resultant set of policy links.vbs
enumerate Resultant set of policy group.vbs
enumerate Resultant set of policy audit policies .vbs
enumerate Resultant set of policy administrative .vbs
enumerate Resultant security settings numeric.vbs
enumerate Resultant security settings boolean.vbs
enumerate Resultant scopes.vbs
enumerate Resultant event log1.vbs
enumerate Resultant event log.vbs
Enumerate Registry Values and Types.vbs
enumerate registry properties.vbs
enumerate recovery configuration options.vbs
Enumerate Published Shares.vbs
enumerate program groups.vbs
enumerate progIDs.vbs
enumerate processor information.vbs
Enumerate Printer Port Properties.vbs
Enumerate Printer Drivers.vbs
Enumerate Printer Connections.vbs
Enumerate Printer Capabilities.vbs
enumerate port connector.vbs
enumerate pointing device properties.vbs
enumerate plug and play signed drives.vbs

enumerate plug and play devices.vbs
enumerate physical memory.vbs
Enumerate Physical Disk Properties.vbs
enumerate parallel port.vbs
Enumerate Page File Properties.vbs
enumerate onboard devices.vbs
Enumerate NTFS Properties.vbs
Enumerate Network Shares.vbs
enumerate memory devices.vbs
Enumerate Logical Disk Drive Properties.vbs
enumerate keyboard.vbs
enumerate IRQ settings.vbs
enumerate internet explorer security zone settings.vbs
enumerate internet explorer security summary settings.vbs
enumerate internet explorer LAN settings .vbs
enumerate installed software features .vbs
enumerate installed software .vbs
enumerate installed or advertised components .vbs
enumerate installed hot fixes .vbs
Enumerate Indexing Service Scopes.vbs
Enumerate Indexing Service Catalogs.vbs
Enumerate Folders Using Dates.vbs
Enumerate Folder Properties.vbs
Enumerate Folder Attributes.vbs
Enumerate Floppy Drives.vbs
Enumerate File Properties.vbs
Enumerate File Attributes.vbs
Enumerate Domain Information for Trust Partners.vbs
enumerate dma channel information .vbs
Enumerate Disk Space by User Using Disk Quotas.vbs
Enumerate Disk Quotas.vbs
Enumerate Disk Quota Settings.vbs
Enumerate Disk Partition Properties.vbs
Enumerate Disk Drive Properties Using FSO.vbs
Enumerate Dfs Targets.vbs
Enumerate Dfs Nodes.vbs
enumerate device memory addresses .vbs
enumerate desktop settings.vbs
enumerate desktop monitor.vbs
enumerate dcom applications .vbs
enumerate dcom application settings .vbs
enumerate computer startup options .vbs
enumerate computer startup commands .vbs
enumerate computer bus.vbs
enumerate computer baseboard.vbs
enumerate computer accounts in active directory .vbs
enumerate computer account attributes .vbs
enumerate component categories .vbs

enumerate classic com class settings.vbs
enumerate classic com class .vbs
Enumerate Available Disk Space.vbs
enumerate allowable video controller.vbs
Enumerate All the Folders on a Computer.vbs
Enumerate All the Files on a Computer.vbs
Enumerate All the Files in a Folder.vbs
enumerate all shadow copy storage area.vbs
enumerate all shadow copies on a computer.vbs
Enumerate All Published Printers.vbs
Enumerate Administrative Tools.vbs
Enumerate Active Directory Database Replication Partners.vbs
Enumerate a Specific Set of Folders.vbs
Enumerate a Specific Set of Files.vbs
enterprise_destination.vbs
Ensure that an Account Will Not Expire.vbs
Ensure that All Drives are Ready.vbs
Enabling Remote Control Settings for a User Account.vbs
Enabling Disk Quotas on the Local Computer.vbs
Enabling DHCP.vbs
Enabling a User to Logon at Any Time.vbs
enableDHCP.vbs
ENABLECHECKPOINTS.VBS
Enable WINS for All Network Adapters.vbs
Enable Terminal Services.vbs
Enable Terminal Services Session Directory.vbs
Enable Terminal Services Password Prompt.vbs
Enable Terminal Services Active Desktop.vbs
Enable Terminal Service Connections.vbs
Enable Single Session Terminal Service Sessions.vbs
Enable PMTU Discovery on all Network Adapters.vbs
Enable IPSec on a Network Adapter.vbs
Enable IPFilter Security on All Network Adapters.vbs
Enable Forcible Terminal Services Logoff.vbs
Enable DNS on All Network Adapters.vbs
Enable Disk Quotas.vbs
Enable Dead Gateway Detection for All Network Adapters.vbs
Enable a UserAccountControl Flag.vbs
Enable a User Account.vbs
enable a global catalog server.vbs
enable a full system restore.vbs
drives1.vbs
drives.vbs
domainname.vbs
dnsserver.vbs
dnsrecord. vbs
DnsImport.vbs
dnsdomainname.vbs

disptree.vbs
dispnode.vbs
DisplayProperty.vbs
Display User Account Password Attributes.vbs
Display the Six IADs Properties of a Domain Object.vbs
Display Tabular Output in a Command Window.vbs
Display Real Time Events in a Command Window.vbs
Display Password Property Attributes.vbs
Display One List of all Attributes that are Single Valued a.vbs
Display Domain Password Attributes.vbs
dispatcher1.vbs
Dispatcher.vbs
Dismount a Volume.vbs
Disabling the 'Smartcard Required' Attribute for a User Account.vbs
Disabling a Local User Account.vbs
DisableScheduledContentDownloads.vbs
disableDHCP.vbs
Disable the User Cannot Change Password Option.vbs
disable full system restore.vbs
Disable a User Account.vbs
Disable a Password Flag.vbs
Disable a global catalog derver.vbs
DFoot.vbs
DevStu.vbs
Determining Whether Attributes are Indexed and in the Globa.vbs
Determining Whether an Account Exists in a Windows NT 4.0 Domain.vbs
Determining When a User Account Expires.vbs
Determining UTC Time.vbs
determining time zone offset from Greenich mean time.vbs
Determining the Primary Group for a User Account.vbs
Determining the Parent Class of an Active Directory Object.vbs
Determining the Owner of a User Account.vbs
Determining the Active Directory Class Type for an Object.vbs
Determining System Uptime.vbs
Determining Other Groups a Group Belongs To.vbs
Determining Local Time on a Computer.vbs
Determining if an Attribute is Operational.vbs
Determining if an Attribute is in the Global Catalog.vbs
Determining if a Domain Controller is in a Site.vbs
Determining Account Expiration Dates for a Windows NT 4.0 Domain.vbs
Determining a Computer's IP Address.vbs
Determine When an Account Expires.vbs
Determine When a Password Was Last Set.vbs
Determine When a Password Expires.vbs
Determine User Account Status.vbs
determine the UTC on a computer.vbs
Determine the Protocols Over Which the Bridgehead Server Re.vbs
Determine the Number of Items in a Dictionary.vbs

determine the local time on a computer.vbs
Determine Process Ownership.vbs
Determine Printer Port Availability.vbs
Determine Logon Hours.vbs
Dependency.vbs
demo.vbs
DELUSERS3.VBS
DELUSERS2.VBS
DELUSERS1.VBS
DELUSERS.VBS
Deleting Published Certificates from a User Account.vbs
Deleting an OU.vbs
Deleting an Active Directory Subnet.vbs
Deleting a User from a Local Group.vbs
Deleting a User Account from Active Directory.vbs
Deleting a Telephone Number from a User Account.vbs
Deleting a Post Office Box from a User Account.vbs
Deleting a Local User Account.vbs
Deleting a Local Group.vbs
Deleting a Group from Active Directory.vbs
Deleting a Calling Station ID from a User Account.vbs
deleteservice.vbs
DeleteNode.vbs
Delete_URL.vbs
Delete Unused Printer Ports.vbs
Delete Specific Printers.vbs
Delete Registry Values.vbs
Delete Print Jobs.vbs
Delete Folders.vbs
Delete Attributes.vbs
delete all shadow copy storage area.vbs
delete all shadow copies on a computer.vbs
Delete All Scheduled Tasks.vbs
Delete All Printers on a Print Server.vbs
Delete All Files in a Folder.vbs
Delete a User Account.vbs
Delete a Terminal Services Account.vbs
Delete a Terminal Service Direct Connect License Server.vbs
Delete a Scheduled Task.vbs
Delete a Registry Key.vbs
Delete a Record from a Recordset.vbs
Delete a Published Folder.vbs
Delete a Printer.vbs
Delete A Printer Port.vbs
Delete a Phone Number.vbs
Delete a Network Share.vbs
Delete a Folder.vbs
Delete a File.vbs

Delete a Disk Quota Entry.vbs
delete a computer account.vbs
delcst1.vbs
delcst.vbs
del_user.vbs
defragram.vbs
Defragment a Volume.vbs
DB.vbs
datetime.vbs
Data1.vbs
Data.vbs
cwsc11.vbs
cwsc1.vbs
CRUSERS9.VBS
CRUSERS8.VBS
CRUSERS.VBS
CROU7.VBS
CROU6.VBS
CROU5.VBS
CROU4.VBS
CROU3.VBS
CROU2.VBS
CROU1.VBS
CROU.VBS
Creating an OU.vbs
Creating an OU in an Existing OU.vbs
Creating an Active Directory Subnet.vbs
Creating an Active Directory Site.vbs
Creating an Active Directory Site Link.vbs
Creating a User, a Group, and an OU.vbs
Creating a Universal Security Group.vbs
Creating a Universal Distribution Group.vbs
Creating a Local User Account.vbs
Creating a Global Security Group.vbs
Creating a Global Group.vbs
Creating a Global Distribution Group.vbs
Creating a Domain Local Security Group.vbs
Creating a Domain Local Distribution Group.vbs
Creating a Contact in Active Directory.vbs
Creating 1,000 User Accounts.vbs
CreateWebVirtualDir.vbs
CreateWebSite.vbs
CreateVirtualWebserverpool.vbs
CreateVirtualWebserver.1vbs.vbs
CreateVirtualWebDir1vbs.vbs
CreateVirtualWebDir.vbs
CreateVirtualFTPserver.vbs
CreateVirtualFTPDir.vbs

CreateUser5.vbs
CreateUser.vbs
CreateImportScript.vbs
CreateFtpVirtualDir.vbs
Create User Account.vbs
Create Unique File Names for Event Log Backups.vbs
Create String and DWORD Values.vbs
Create Script Documentation Using Script Comments.vbs
Create New Folders.vbs
Create Expanded String Values.vbs
Create and Name a Text File.vbs
Create an IPv6 Address (AAAA) DNS Record.vbs
Create an Instance of Internet Explorer.vbs
Create an ATM Address to Name (ATMA) DNS Record.vbs
Create an Andrew File System Database Server DNS Record.vbs
Create a Well-Known Services (WKS) DNS Record.vbs
Create a Text File.vbs
Create a Text (TXT) DNS Record.vbs
create a system restore point.vbs
create a shadow copy.vbs
create a shadow copy storage area.vbs
Create a Route Through (RT) DNS Record.vbs
Create a Responsible Person (RP) DNS Record.vbs
Create a Registry Key.vbs
Create a Process on a Remote Computer.vbs
Create a Process in a Hidden Window.vbs
Create a Primary DNS Zone.vbs
Create a Permanent Event Filter.vbs
Create a Permanent Event Consumer.vbs
Create a Network Share.vbs
Create a Name Server DNS Record.vbs
Create a MultiString Value.vbs
Create a Mailbox Rename (MR) DNS Record.vbs
Create a Mailbox (MB) DNS Record.vbs
Create a Mail Information (MI) DNS Record.vbs
Create a Mail Group (MG) DNS Record.vbs
Create a Mail Forwarding Agent (MF) DNS Record.vbs
Create a Mail Exchanger (MX) DNS Record.vbs
Create a Mail Agent for Domain (MD) DNS Record.vbs
Create a Local Group on a Computer.vbs
Create a Host Address (A) DNS Record.vbs
Create a Higher Priority Process.vbs
Create a Folder.vbs
Create a Dfs Node.vbs
Create a Custom Event Log.vbs
create a computer account.vbs
create a computer account for a user.vbs
cr_user.vbs

COUNT.vbs
copyname1.vbs
copyname.vbs
Copying an Active Directory User Account.vbs
copying an active directory computer account.vbs
Copying Allowed Logon Hours from One Account to Another.vbs
Copying a Published Certificate to a User Account.vbs
Copy Previous Days Event Log Events to a Database.vbs
Copy Folders Using WMI.vbs
Copy Folders Using the Shell Object.vbs
Copy Event Log Events to a Database.vbs
Copy a Set of Files.vbs
Copy a Folder.vbs
Copy a File.vbs
convname1.vbs
convname.vbs
Converting WMI Date-Time Values.vbs
contweb.vbs
contsrv.vbs
contftp.vbs
Contacts.vbs
ConstructLAT.vbs
Connect to an ADO Database.vbs
Configuring User Account Telephone Numbers.vbs
Configuring the Expiration Date for a User Account.vbs
Configuring Terminal Services Session Properties for a User.vbs
Configuring Terminal Services Profile Properties for a User.vbs
Configuring Terminal Services Environment Properties for a .vbs
Configuring and Operating System Restore.vbs
Configuring a Static IP Address.vbs
Configure Zero-Broadcast Use for All Network Adapters.vbs
configure WMI settings.vbs
Configure Trust Relationship Properties.vbs
Configure the WINS Server for a Network Adapter.vbs
Configure the Number of Allowed Forward Packets.vbs
Configure the MTU for all Network Adapters.vbs
Configure the Keep Alive Interval for all Network Adapters.vbs
Configure the IPX Virtual Network Number.vbs
Configure the IPX Frame Type.vbs
Configure the IP Connection Metric for a Network Adapter.vbs
Configure the Indexing Service to Autostart.vbs
Configure the IGMP Level for All Network Adapters.vbs
Configure the Gateways for a Network Adapter.vbs
Configure the Forward Buffer Memory for All Network Adapters.vbs
Configure the DNS Suffix Search Order for All Network Adapters.vbs
Configure the DNS Server Search Order for a Network Adapter.vbs
Configure the DNS Domain for a Network Adapter.vbs
configure the default WMI namespace.vbs

Configure Terminal Services Time Zone Redirection.vbs
Configure Terminal Services Session Time Limit.vbs
Configure Terminal Services Session Directory Location.vbs
Configure Terminal Services Properties for a User.vbs
Configure Terminal Services Policy Property.vbs
Configure Terminal Services Mode.vbs
Configure Terminal Services Home Directory.vbs
Configure Terminal Services Encryption Level.vbs
Configure Terminal Services Color Depth.vbs
Configure Terminal Services Color Depth Policy.vbs
Configure Terminal Services Client Wallpaper Settings.vbs
Configure Terminal Services Client Settings.vbs
Configure Terminal Services Client Connection Settings.vbs
Configure Terminal Services Broken Connection Settings.vbs
Configure Terminal Service Remote Control Settings.vbs
configure system startup delay.vbs
Configure Printer Priority.vbs
Configure Printer Locations.vbs
Configure Printer Availability.vbs
Configure Page File Properties.vbs
Configure Organization Properties for a User Account.vbs
Configure NetBIOS Use on a Network Adapter.vbs
Configure Home Networking Firewall Logging Settings.vbs
Configure Explicit Terminal Services Logon.vbs
Configure Event Log Properties.vbs
Configure Dynamic DNS Registration for a Network Adapter.vbs
Configure Disk Quota Settings.vbs
Configure ARP Queries to Use Source Routing.vbs
Configure ARP Queries to Use EtherSNAP.vbs
conduct a system restore.vbs
Compress Folders.vbs
Colors.vbs
Clone.vbs
CLOCK.vbs
Clearing the General Properties of an OU.vbs
Clearing the COM+ Partition Link Set of an OU.vbs
Clearing Terminal Services Properties for a User Account.vbs
Clearing Department and Direct Report Information from a Us.vbs
Clearing COM+ Attributes from a User Account.vbs
Clearing All Published Certificates from a User Account.vbs
Clearing Address Page Information for a User Account.vbs
cleareventlog.vbs
Clear User Account Address Attributes.vbs
Clear the Group Policy Links Assigned to an OU.vbs
Clear Telephone Properties for a User Account.vbs
Clear Telephone Attributes.vbs
Clear DNS Server Cache.vbs
Clear Attributes.vbs

Clear a Database Table.vbs
cleannts1.vbs
cleannts.vbs
classpath.vbs
class_FormBuilder.vbs
Check the Size of a File Before Reading It.vbs
Check Registry Key Access Rights.vbs
CHART3.VBS
CHART1.VBS
CHART.VBS
Changing the Scope of a Group.vbs
Changing the Manager of an OU.vbs
Changing the Local Administrator Password.vbs
ChangeUserProperty.vbs
ChangeUserPassword.vbs
ChangeProperty.vbs
Change Volume Names.vbs
Change User Password.vbs
Change User Account Attributes.vbs
Change the Priority Of a Running Process.vbs
Change the Drive Letter of a Volume.vbs
Change Print Job Start Time.vbs
Change Print Job Priority.vbs
Change Folder Attributes.vbs
Change File Extensions.vbs
Change DNS Zone Type.vbs
change computer account attributes.vbs
chaccess.vbs
Celsju.vbs
CDir.vbs
CConnect.VBS
CCLogoff.vbs
Calculator.vbs
CacheSettings.vbs
build2.vbs
build1.vbs
build.vbs
BlockNimba.vbs
Bind to a Specific Disk Drive.vbs
BASICDS.vbs
Basic1.vbs
BASIC.vbs
Baseline Performance Monitoring.vbs
backupeventlog.vbs
Backup and Clear Large Event Logs.vbs
Backup and Clear an Event Log.vbs
Asynchronously Enumerate Files.vbs
Asynchronous Event Log Query.vbs

Associating Disk Partitions with Physical Disk Drives.vbs
Assigning a Published Certificate to a User Account.vbs
Assigning a New Group Policy Link to an OU.vbs
Assigning a Group Manager.vbs
Assign Terminal Services Initial Program.vbs
apver1.vbs
apver.vbs
ApplicationFilterList.vbs
Appending Address Page Information for a User Account.vbs
Appending a Home Phone Number to a User Account.vbs
Append a Phone Number.vbs
Append a MultiValued Attribute.vbs
APManPrj.vbs
ANIMATEDPROPERTIES.vbs
ANIFILT.vbs
Analyze Volume Defragmentation.vbs
AgentWait.vbs
Agent.vbs
Age All DNS Records.vbs
afstrnld.vbs
afstrenu.vbs
aform.vbs
adsutil5.vbs
adsutil4.vbs
adsutil1.vbs
adsutil.vbs
ADDUSERS13.VBS
ADDUSERS12.VBS
ADDUSERS1.VBS
ADDUSERS.VBS
ADDUSER.VBS
AddScheduledContentDownload.vbs
AddLATEntry.vbs
AdditionalKey.vbs
Adding New Members to a Group.vbs
Adding 'Command Prompt Here' to Windows Explorer.vbs
Adding a User to Two Security Groups.vbs
Adding a User to a Local Group.vbs
Adding a Route to the Dial-In Properties of a User Account.vbs
Adding 1,000 Users to a Security Group.vbs
AddDeniedMethod.vbs
addclass.vbs
dublicated.vbs
Add WMI Data to an Event Log Entry.vbs
Add Elements to a Dictionary.vbs
Add an Indexing Service Scope.vbs
Add an Indexing Service Catalog.vbs
Add a Volume Mount Point.vbs

Add a Terminal Services Account.vbs
Add a Template to the Windows Explorer New Menu.vbs
Add a Support URL to an Event Log Entry.vbs
Add a Printer Connection.vbs
Add a New Record to a Database.vbs
Add a Disk Quota Entry.vbs
activate windowsonline.vbs
activate windowsoffline.vbs
Activate Windows Online.vbs
act_samp.vbs
6.vbs
5.vbs
4.vbs
3B.VBS
3.vbs
2.vbs
1.vbs
01-logon.vbs
Add_DOD.vbs