Cyber Security Challenges: Designing Efficient Intrusion Detection Systems and Antivirus Tools

Srinivas Mukkamala, Andrew Sung and Ajith Abraham[*]

Department of Computer Science, New Mexico Tech, USA
[*]School of Computer Science and Engineering, Chung-Ang University, Korea
Email: ajith.abraham@ieee.org,

Several information security techniques are available today to protect information and systems against unauthorized use, duplication, alteration, destruction and virus attacks. Intrusion detection a key component of information security (protect, detect and react) and network defense, provides information on successful and unsuccessful attempts to compromise information assurance (availability, integrity, and confidentiality). Intruders can broadly be categorized into two types: external intruders, who are unauthorized users of information, and systems they attack, and internal intruders, who have permission to access information and systems with a few restrictions. In this chapter we first present the state-of-the-art of the evolution of intrusion detection technology and address a few intrusion detection techniques and IDS implementations. An overview of computer attack taxonomy and computer attack demystification along with a few detection signatures is presented. Special emphasis is also given to the current IDS limitations. Further we describe few obfuscation techniques applied to recent viruses that were used to thwart commercial grade antivirus tools.

1. Introduction to Intrusion Detection Systems

Intrusion detection an important component of information security technology helps in discovering, determining, and identifying unauthorized use, duplication, alteration, and destruction of information and information systems. Intrusion detection relies on the assumption that information and information systems under attack exhibit several distinguishable behavioral patterns or characteristics to that of the normal ones. Though intrusion detection technology is becoming ubiquitous in current network defense; it lacks basic definitions and mathematical understanding. Intrusion detection being subjective; each Intrusion Detection System (IDS) has a different classification and attack labeling mechanisms. It is most common for IDSs to alarm on any set of known attack behaviors. In the due course of determining whether a particular activity is normal or malicious, IDS fail to alarm an attack (false negative) or alarm normal activity as malicious (false positive).

The most popular way to detect intrusions has been done by using audit data generated by operating systems and by networks. Since almost all activities are logged on a system, it is possible that a manual inspection of these logs would allow intrusions to be detected. It is important to analyze the audit data even after an attack has occurred, for determining the extent of damage occurred, this analysis helps in attack trace back and also helps in recording the attack patterns for future prevention of such attacks. An intrusion detection system can be used to analyze audit data for such insights. This makes intrusion detection system a valuable real-time detection and prevention tool as well as a forensic analysis tool.

2. A Review on Intrusion Detection Systems

An intrusion is an activity or a sequence of activities that result in a compromise or intend to compromise the aspects of information assurance. Intrusion detection is a security technology of great significance to critical infrastructure protection that attempts to detect, and respond to intrusions against information and information systems. IDSs that rely on audit trails for deciding whether a particular activity is intrusive or not; compliments other security technologies (firewalls, file integrity checkers, vulnerability scanners, and anti virus tools). IDS also provide information for forensic analysis and to detect non-repudiation activities based on the audit trails collected. IDS that detect intrusions based on deviations from normal to abnormal state using user or systems profiles is defined as anomaly detection. Anomaly detection tends to detect novel attacks with the expense of false positives. Signatures are a set of actions, conditions or activities, when met indicate an intrusion. IDS that rely on signatures are defined as misuse or signature based detection systems. Misuse detection systems tend to higher detection rate with the expense of false negatives.

2.1 Intrusion Detection Models

Though there are several IDS implementations most of them share the common task of detecting intrusions based on audit trails or system state. These components include data collection module, analysis module, storage module, and response module. Most of the IDS are a software package and/or hardware, or part of a larger system. IDSs are built with a set of components that together define an IDS model. A generic model of IDS is illustrated in Figure 1.

Data collection module provides information to the rest of the system to decide whether a particular activity is intrusive or not. Data collection module collects audit trails (user logs, network trails, system calls, etc.) for the other IDS components to make decisions, with out this module the IDS becomes un-functional. An important issue in the data collection module is audit data reduction. Instead of passing the raw data in great detail to the analysis module to decide whether a particular activity is malicious or normal, designers implement systems that eliminate audit information believed to be unimportant for intrusion analysis. The goal of audit reduction might not be limited to passing important, reduced or summarized audit trails to the analysis module; they also help in reducing the complexity of the analysis module.

Analysis module analyses inputs (audit trails) form the data collection module. Large chunk of the IDS research is concentrated on creating novel classifiers in terms of better IDS performance (faster classification, low false alarms and higher accuracies). Several analysis techniques are being proposed ranging from statistical analysis, pattern matching, machine learning, file integrity checkers and artificial immune system methods. Analysis module helps in automated analysis of data by reducing human intervention and speeds up the process of identifying intrusions in real time.

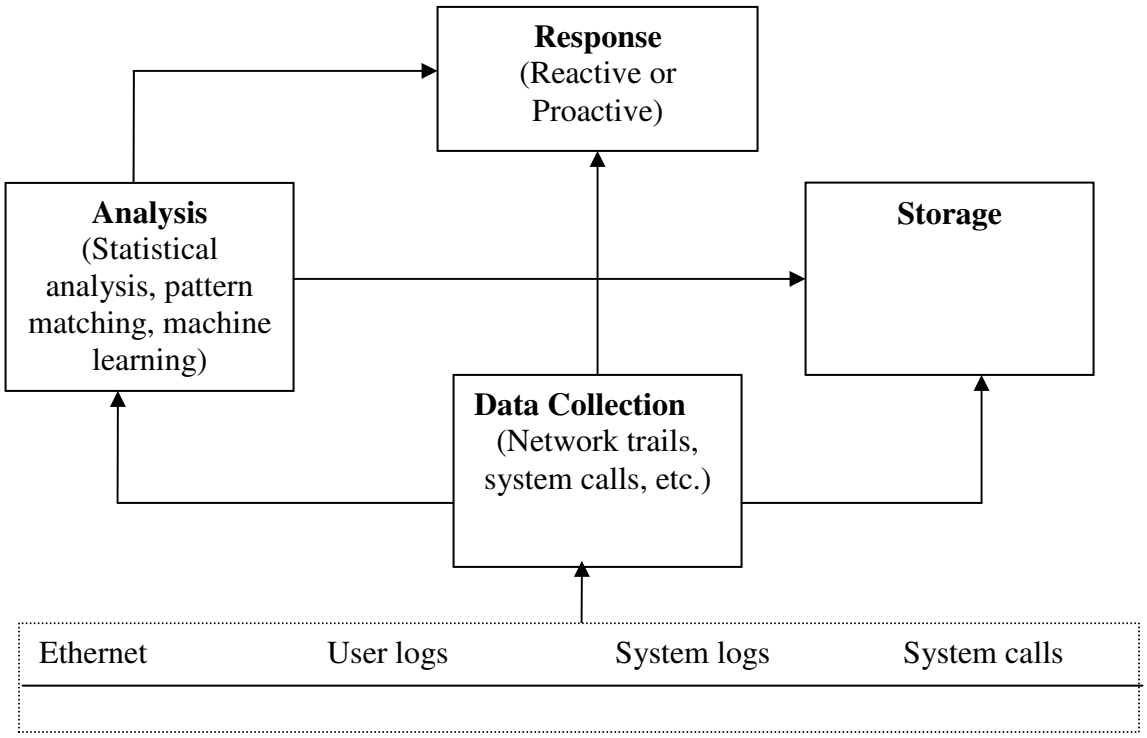

Figure 1 Generic Intrusion Detection System Model

Storage module provides a mechanism to store data collected by data collection and analysis modules in a secure fashion. Data stored might be used for building new signatures, updating user and system profiles, forensics analysis, and identifying key audit information.

Response module can be designed in an active or a proactive mode. Most of the current IDS are designed to be proactive; they set an alarm when an intrusion takes place. Leap forward technology in the field of IDSs is designing and implementing them as reactive devices rather than an aftermath device. Intrusion detection prevention systems (IDPS) not only spot intrusions but also intercept and stop intrusions.

2.1.1 Signature Based or Misuse Intrusion Detection

The idea of misuse detection is to represent attacks in the form of a pattern or a signature so that the same attack can be detected and prevented in future [1]. These systems can detect many or all known attack patterns, but they are of little use for detecting naive attack methods. The main issues of misuse detection is how to build signatures that include possible signatures of attacks and all possible variations of the pertinent attack to avoid false negatives, and how to build signatures that do not match non-intrusive activities to avoid false positives.

2.1.2 Anomaly Detection

The idea here is that if we can establish a normal activity profile for a system, in theory we can flag all system states varying from the established profile as intrusion attempts [2]. However, if the set of intrusive activities is not identical to the set of anomalous activities, the situation becomes more interesting instead of being exactly the same, we find few interesting possibilities.

Anomalous activities that are not intrusive are flagged as intrusive, though they are false positives. Actual intrusive activities that go undetected are called false negatives. This is a serious issue, and is far more serious than the problem of false positives. One of the main issues of anomaly detection systems is the selection of threshold levels so that neither of the above problems is unreasonably magnified. Anomaly detection is usually computationally expensive because of the overhead of keeping track of and possibly updating several system profiles.

3. Computer Attack Taxonomy

Good attack taxonomy makes it possible to classify individual attacks into groups that share common properties. Taxonomy should hold the property; if classifying in one category excludes all others because categories do not overlap. Good attack taxonomy should have the following characteristics:

- Mutually exclusive: categories should not over lap. Classifying an attack into one category excludes it from others.
- Exhaustive: an attack classified into a category includes all possibilities.
- Unambiguous: should be applied to all systems irrespective of what is used or who is classifying. It should be clear and precise so that classification does not become uncertain.
- Repeatable: repeated applications result in the same classification, regardless of who is classifying.
- Acceptable: should be logical and intuitive so that they become generally approved.

Network and computer elements identifies the specific elements of software, hardware, or protocols where the attacks take place. The view of probe and Denial of Service (DoS) characterization focuses attention on specific and tangible, as opposed to logical, elements of a system, protocol, or data packet that allows the adversary to achieve their goal, of reconnaissance and denial of some service. This entity may not be the service that the attacker wishes to compromise, but is the element of the network which will allow their ultimate purpose to succeed. Given that a probe and DoS attack can consist of multiple exploits, there exist multiple targets spread out over time and space. With this view of the attack space, we recognize that adversaries identify and attack real elements of networks and computers for compromise, whether hardware, software, or firmware. Services mapped to Open Systems Interconnection (OSI) and the TCP/IP models are given on Figure 2. The OSI Model includes two layers not often distinguished in a communication process, i.e. the presentation and session layers, while the TCP/IP Model only utilizes four levels of granularity. OSI Probe and DoS attacks view with specific to network and computer elements are given in Figure 3.

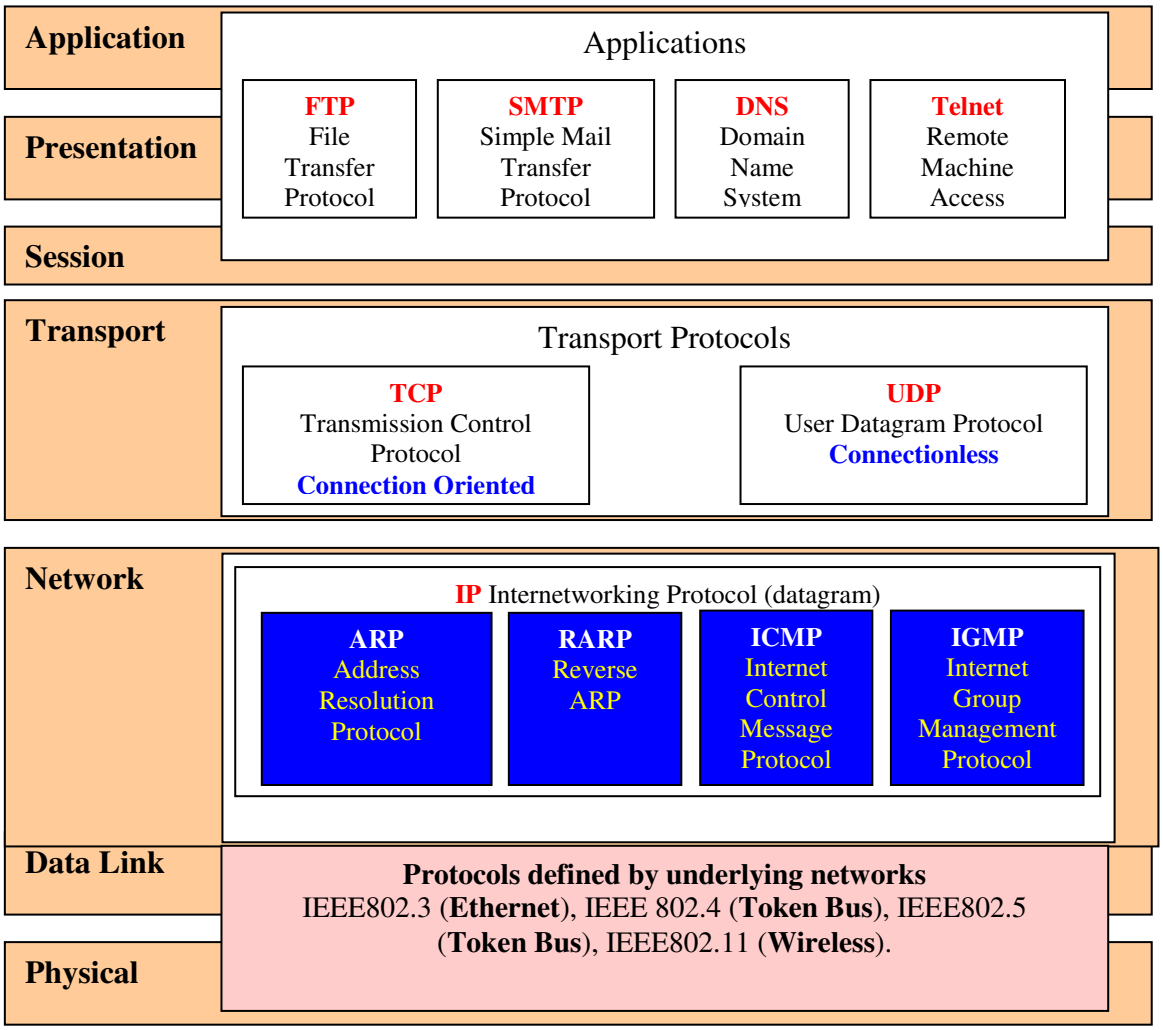

Figure 2. Layered Approach to Network and Computer Elements

3.1. Probing

Probing is a class of attacks where an attacker scans a network to gather information or find known vulnerabilities. An attacker with a map of machines and services that are available on a network can use the information to look for exploits. There are different types of probes: some of them abuse the computer's legitimate features; some of them use social engineering techniques. This class of attacks is the most commonly heard and requires very little technical expertise. Different types of probe attacks are illustrated in Table 1 [3,4].

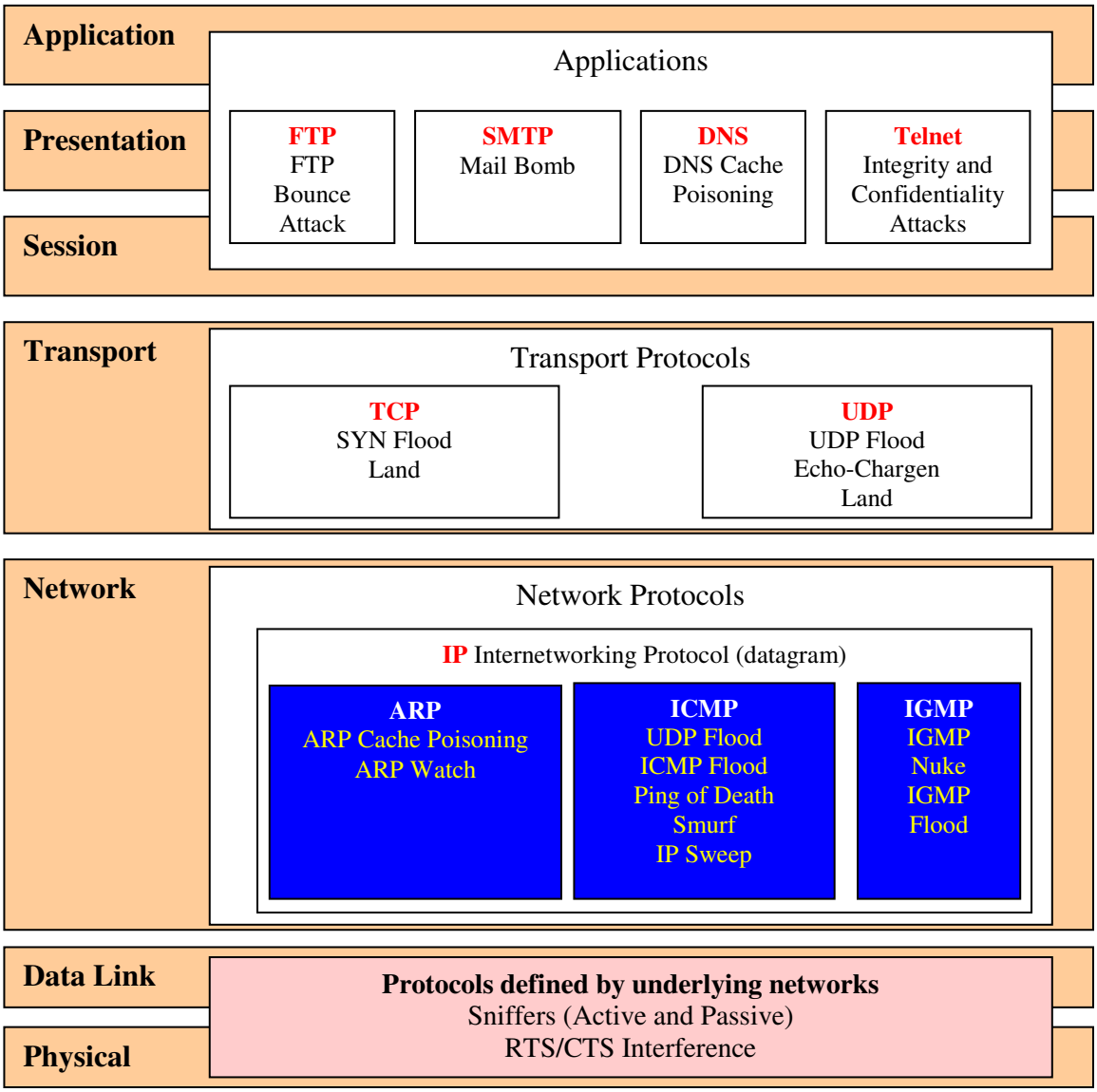

Figure 3. Layered Attack Views to Network and Computer Elements

i. **Ipsweep:** Probing attack is performed against all operating systems using ICMP service where an adversary performs a surveillance sweep to determine which hosts are responding on a network. Information obtained from surveillance is useful to an adversary in launching automated attacks or in making the vulnerable hosts as stepping stones for future distributed attacks. This attack helps the adversary identify active machines on the network and might degrade services for legitimate users. Looking for multiple ping requests, destined for all possible machines on a network, all coming form the same host can help detect this attack.

ii. **Mscan:** Probing tool used to perform an attack against all operating systems using multiple services; where an adversary uses both DNS zone transfers and/or brute force scanning of IP addresses to locate machines, and look for vulnerabilities to launch future attacks. This attack helps the adversary identify known vulnerabilities on the network and the host machine. Looking for connection requests form an out side machine to vulnerable services (netbios-ns, epmap, ms-sql-m, dameware, microsoft-ds, realsecure, domain, bind, imap, pop, NFS, cgi-bin, open X servers) within a specified period of time, can help detect this attack.

iii. **Nmap:** General-purpose probing tool used to perform network scans against all operating systems using multiple services with user specified time intervals; an adversary can specify which services to scan for, how much time to wait between each service, and whether the services should be scanned sequentially or in a random order. This attack helps the adversary identify services running, operating system, and known vulnerabilities on the network and the target machine. Looking for connection requests to multiple services within a specific time window, can help detect this attack.

iv. **SAINT:** Security Administrator's Integrated Network Tool is used to gather information about remote hosts (all operating systems) using multiple services; an adversary uses a few network services such as finger, ftp, tftp, statd, rpc, NIS, NFS and other relevant network services. This attack helps the adversary identify network services running, system flaws, and critical security flaws on the victims' machine. Looking for connections requests to specific network services form a machine other than an authorized machine within in a specific time window, can help detect this attack.

v. **Satan:** Probing tool used to perform scans against all operating systems using a few network services; where an adversary uses legitimate network services to gather information on particular vulnerabilities on the victims' machine. Looking for connections requests to specific vulnerable network services form a machine other than an authorized machine within in a specific time window, can help detect this attack.

vi. **SYN Stealth Scan:** Probing attack performed against all operating systems using multiple TCP services where an adversary performs surveillance to determine which hosts are responding to specific services on a network. Information obtained form surveillance is useful to an adversary in launching automated attacks, in making the vulnerable hosts as stepping stones for future distributed attacks or for launching future denial of service attacks. This attack helps the adversary identify active machines on the network and might degrade services for legitimate users. Looking for multiple half open TCP connection requests, destined for all possible machines on a network, can help detect this attack.

vii. **FIN Stealth Scan:** Probing attack performed against all operating systems except Windows 95/NT; when SYN scanning isn't clandestine enough. By theory closed ports are required to reply to a probe packet with a RST, while open ports must ignore the packets. An adversary abuses the feature; to determine what services are running on a network or a host system. This scan bypasses the traditional firewalls and network filters. This attack helps the adversary identify active services and the hosts' operating system. Looking for connection requests to closed services within in a specific time window, can help detect this attack.

viii. **Ping Sweep:** Snooping performed against all operating systems using ICMP where an adversary performs a surveillance sweep to determine which hosts are responding on a network. Information obtained from surveillance is useful to an adversary in launching

automated attacks or in making the vulnerable hosts as stepping stones for future distributed attacks. This attack helps the adversary identify active machines on the network and might degrade services for legitimate users. Ping sweep if repeated continuously or launched in a coordinated fashion might result into a low level DoS attack. Looking for multiple ping requests, destined for all possible machines on a network, all coming form the same host or within a specific time window from multiple hosts, can help detect this attack.

ix. **UDP scan:** Probing attack performed against all operating systems using UDP where an adversary sends 0 byte UDP packets to each UDP service on the target machine to determine which services are running on the victims' machine. This attack helps the adversary identify vulnerable UDP services on victims' network. This information is mostly used to launch automated distributed and coordinated denial of service attacks. Looking for multiple UDP packets with 0 bytes within a specific time window, can help detect this attack.

x. **Null Scan:** Probing attack performed against all operating systems except Window 95/NT where an adversary turns off all flag options (FIN, URG, PUSH, etc.). This attack helps identify victims' operating system; by sending connection requests to services running on the host machine. Looking for multiple connection requests with all the flags turned off within a specific time window, destined for all possible machines on a network, can help prevent this attack.

xi. **IP scan:** Snooping performed against all operating systems using raw IP packets without any specified future protocol header. An adversary sends raw IP packets without any specific future protocol header to each specific protocol on the victims' machine. If an ICMP message stating protocol unreachable is received, then it's assumed that specific protocol is not in use. This attack helps identify all the supported protocols on a victims' network. Looking for multiple connection requests without a specific service within a specific time, can help detect this attack.

xii. **ACK scan:** Snooping attack performed to map firewall rule sets where an adversary sends ACK packets (random acknowledgement/sequence numbers) to specific ports. If RST comes back, the specified port is classified as "unfiltered". If nothing comes back or an ICMP error message comes, the specified port is classified as "filtered". This attack helps identify filtered services and a type of firewall a victims' network has. Looking for Random ACK packets, can help detect this attack.

xiii. **Window scan:** Probing attack performed against all operating systems using the vulnerability in TCP window size reporting. This attack helps the adversary identify active services as well as filtered services on a victims' machine.

xiv. **RCP scan:** Snooping performed against all operating systems using multiple services to identify active remote procedure call services. This attack helps the adversary identify active remote procedure call services as well as the associated program and version numbers. This information is mostly used to execute arbitrary code by the adversary on a victims' machine. Looking for multiple connection requests to specific remote procedure call services within a specific time, can help detect this attack.

Table 1. Different Types of Probe Attacks

Attack Type	Service	Mechanism	Effect of the attack
Ipsweep	ICMP	Abuse of feature	Identifies active machines
Mscan	Many	Abuse of feature	Looks for known vulnerabilities
Nmap	Many	Abuse of feature	Identifies active ports on a machine
Saint	Many	Abuse of feature	Looks for known vulnerabilities
Satan	Many	Abuse of feature	Looks for known Vulnerabilities
SYN Stealth	Multiple	Abuse of feature	Identifies active machines
FIN Stealth	Multiple	Abuse of feature	Identifies active services
Ping Sweep	ICMP	Abuse of feature	Identifies active machines
UDP Scan	Multiple	Abuse of feature	Identifies active UDP services
Null Scan	Multiple	Abuse of feature	Identifies active services
IP Scan	Multiple	Abuse of feature	Identifies active protocols
ACK Scan	Multiple	Abuse of feature	Identifies the firewall mechanism (stateful or simple network filter)
Window Scan	Multiple	Mis-configuration	Identifies active services
RCP Scan	Multiple	Abuse of feature	Identifies active remote procedure call ports (RPC)

3.2. Denial of Service Attacks

Denial of Service is a class of attacks where an attacker makes some computing or memory resource too busy or too full to handle legitimate requests, thus denying legitimate users access to a machine. There are different ways to launch DoS attacks: by abusing the computers legitimate features; by targeting the implementations bugs; or by exploiting the system's misconfigurations. DoS attacks are classified based on the services that an attacker renders unavailable to legitimate users. Some of the popular attack types are illustrated in Table 2 [3,4].

i. **Apache2:** DoS attack performed against an apache web server where an adversary submits an http request with several http headers. In theory if the server receives too many of such requests it will slow down the functionality of the web server and eventually crashes. This attack denies the web service temporarily; the service can be regained automatically with system administrator's intervention.

ii. **Back:** DoS attack performed against an apache web server where an adversary submits an URL request with several front slashes. While trying to process these requests, the server's

service becomes unavailable for legitimate users. This attack denies the web service temporarily; the service can be regained automatically.

iii. **Land:** DoS attack performed against TCP/IP implementations where an adversary sends a spoofed SYN packet where the source and destination IP address are the same. In theory it's not possible to have the same destination address as the source address. The adversary targets the badly configured networks and uses the innocent machines as zombies for performing distributed attacks. This attack can be prevented by carefully configuring the network, which prevents requests containing the same source and destination IP addresses.

iv. **Mail bomb:** DoS attack performed against the server where an adversary floods the mail queue, possibly causing failure. The adversary tries to send thousands of mails to a single user. This attack denies the service permanently. The service can be regained by the system administrator intervention; blocking the mails coming from or to the same user within a short period of time can prevent the attack.

v. **SYN Flood (Neptune):** DoS attack performed against every TCP/IP implementations where an adversary utilizes the half open TCP connections to flood the data structure of half open connections on the innocent server causing to deny access to legitimate requests. This attack in some cases can cause permanent failure. The service can be regained automatically. Looking for a number of simultaneous SYN packets coming form the same host or unreachable host in a given short period of time can prevent this attack.

vi. **Ping of Death (PoD):** DoS attack performed against older versions of operating systems where an adversary tries to send an oversized IP packet, and the system reacts in an unpredictable manner, causing crashing, rebooting and even freezing in some cases. This attack causes temporary failure of services. Looking for Internet Control Message Protocol (ICMP) packets that are longer than 64000 bytes and blocking them is the way to prevent this attack.

vii. **Process Table:** DoS attack performed against a variety of different Unix systems where an adversary tries to allocate a new process for every incoming TCP/IP connection; when the systems process table is filled completely, legitimate commands are prevented from being executed. This attack causes temporary failure of services. Looking for large number of active connections on a single port helps in preventing this attack.

viii. **Smurf:** DoS attack performed against all the systems connected to the Internet where an adversary uses the ICMP echo request packets to IP broadcast addresses from remote locations to deny services. This attack causes temporary denial of services and can be automatically recovered. Looking for a large number of echo replies to the innocent machine from different places without any echo request made by the innocent machine helps in detecting this attack.

ix. **Syslogd:** DoS attack performed against Solaris servers where an adversary tries to kill the syslogd service remotely. The adversary exploits the DNS lookup feature, if the source IP address does match the DNS record then the syslogd crashes with a segmentation fault. This attack permanently denies the services and can be recovered with the system administrator intervention.

x. **Teardrop:** DoS attack performed against older versions of TCP/IP stack where an adversary exploits the feature of IP fragment reassembly. This attack denies the services temporally.

xi. **Udpstrom:** DoS attack performed against networks where an adversary utilizes the UDP service feature to cause congestion and slowdown. This attack denies the services permanently and can be resumed with system administrator intervention. This attack can be identified by looking for spoofed packets and inside network traffic.

Table 2. Denial of Service Attacks

Attack Type	Service	Mechanism	Effect of the attack
Apache2	http	Abuse	Crashes httpd
Back	http	Abuse/Bug	Slows down server response
Land	http	Bug	Freezes the machine
Mail bomb	N/A	Abuse	Annoyance
SYN Flood	TCP	Abuse	Denies service on one or more ports
Ping of Death	Icmp	Bug	None
Process table	TCP	Abuse	Denies new processes
Smurf	Icmp	Abuse	Slows down the network
Syslogd	Syslog	Bug	Kills the Syslogd
Teardrop	N/A	Bug	Reboots the machine
Udpstrom	Echo/ Chargen	Abuse	Slows down the network

4. Significant Feature Selection for Intrusion Detection

Feature selection and ranking is an important issue in intrusion detection [5,6]. Of the large number of features that can be monitored for intrusion detection purpose, which are truly useful, which are less significant, and which may be useless? The question is relevant because the elimination of useless features (the so-called audit trail reduction) enhances the accuracy of detection while speeding up the computation, thus improving the overall performance of an IDS. In cases where there are no useless features, by concentrating on the most important ones we may well improve the time performance of an IDS without affecting the accuracy of detection in statistically significant ways. The feature ranking and selection problem for intrusion detection is similar in nature to various engineering problems that are characterized by:

- Having a large number of input variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ of varying degrees of importance to the output **y**; i.e., some elements of **x** are essential, some are less important, some of them may not be mutually independent, and some may be useless or irrelevant (in determining the value of **y**)
- Lacking an analytical model that provides the basis for a mathematical formula that precisely describes the input-output relationship, $\mathbf{y} = \boldsymbol{F}(\mathbf{x})$
- Having available a finite set of experimental data, based on which a model (e.g. neural networks) can be built for simulation and prediction purposes

Due to the lack of an analytical model, one can only seek to determine the relative importance of the input variables through empirical methods. A complete analysis would require examination of all possibilities, e.g., taking two variables at a time to analyze their dependence or correlation, then taking three at a time, etc. This, however, is both infeasible (requiring 2^n experiments!) and not infallible (since the available data may be of poor quality in sampling the whole input space). Features are ranked based on their influence towards the final classification. Description of most important features as ranked by three feature-ranking algorithms (support vector decision function, linear genetic programming, and multivariate adaptive regression splines) is given in Tables 3, 4, and 5 [5].

4.1 SVM Specific Feature Ranking Method

It is of great interest and use to find exactly which features underline the nature of connections of various classes. This is precisely the goal of data visualization in data mining. The problem is that the high-dimensionality of data makes it hard for human experts to gather any knowledge. If we knew the key features, we could greatly reduce the dimensionality of the data and thus help human experts become more efficient and productive in learning about network intrusions.

The information about which features play key roles and which are more neutral is "hidden" in the SVM decision function. The decision function is formulated using linear kernels as follows [7,8].

$$F(X) = <W, X> + b \qquad (1)$$

The point X is predicted to be in class A or "positive class" if the F(X) is positive, and class B or "negative class" if F(X) is negative. We can rewrite the formula (2) to expand the dot product of W and X.

$$F(X) = \Sigma W_i X_i + b \qquad (2)$$

One can see that the value of F(X) depends on the contribution of each factor, $W_i X_i$. Since X_i can take only $b \gtrless 0$, the sign of W_i indicates whether the contribution is towards positive classification or negative classification. The absolute size of W_i measures the strength of this contribution. In other words if W_i is a large positive value, then i^{th} feature is a key factor of "positive class" or class A. Similarly if W_i is a large negative value then i^{th} feature is a key factor of the "negative class" or class B. Consequently the W_i, which is close to zero, either positive or negative, carries little weight. The feature, which corresponds to this W_i, is said to be garbage feature and removing it has very little effect on the classification. Having retrieved this information directly from SVMs' decision function, we rank the W_i, from largest positive to largest negative. This essentially provides the soft partitioning of the features into the key features of class A, neutral features, and key features of class B. We say soft partitioning, as it either depends on a threshold on the value of W_i, which will define the partitions, or the proportions of the features, which we want to allocate to each of the partitions. Both the threshold and the value of proportions can be set by the human expert.

Support Vector Decision Function Ranking:
The input ranking is done as follows: First the original data set is used for the training of the classifier. Then the classifier's decision function is used to rank the importance of the features. The procedure is:

1. Calculate the weights from the support vector decision function;
2. Rank the importance of the features by the absolute values of the weights;

Table 3. Most Important Features Description as Ranked by SVDF

Ranking Algorithm	Feature Description
Normal	▪ destination bytes: number of bytes received by the source host form the destination host ▪ dst_host_count: : number of connections from the same host to the destination host during a specified time window ▪ logged in: binary decision (1 successfully logged in, 0 failed login) ▪ dst_host_same_srv_rate: % of connections to same service ports from a destination host ▪ flag: normal or error status of the connection
Probe	▪ source bytes: number of bytes sent from the host system to the destination system ▪ dst_host_srv_count: : number of connections from the same host with same service to the destination host during a specified time window ▪ count: number of connections made to the same host system in a given interval of time ▪ protocol type: type of protocol used to connect (e.g. tcp, udp, icmp, etc.) ▪ srv_count: : number of connections to the same service as the current connection during a specified time window
DoS	▪ count: number of connections made to the same host system in a given interval of time ▪ srv_count: : number of connections to the same service as the current connection during a specified time window ▪ dst_host_srv_serror_rate: % of connections to the same service that have SYN errors from a destination host ▪ serror_rate: % of connections that have SYN errors ▪ dst_host_same_src_port_rate: % of connections to same service ports from a destination host
U2Su	▪ source bytes: number of bytes sent from the host system to the destination system ▪ duration: length of the connection ▪ protocol type: type of protocol used to connect (e.g. tcp, udp, icmp, etc.) ▪ logged in: binary decision (1 successfully logged in, 0 failed login) ▪ flag: normal or error status of the connection
R2L	▪ dst_host_count: no of connections from the same host to the destination host during a specified time window ▪ service: type of service used to connect (e.g. fingure, ftp, telnet, ssh, etc.) ▪ duration: length of the connection ▪ count: number of connections made to the same host system in a given interval of time ▪ srv_count: : number of connections to the same service as the current connection during a specified time window

Table 4. Most Important Features Description as Ranked by LGPs

Ranking Algorithm	Feature Description
Normal	▪ hot: number of "hot" indicators ▪ source bytes: number of bytes sent from the host system to the destination system ▪ destination bytes: number of bytes received by the source host form the destination host ▪ num_compromised: number of compromised conditions ▪ dst_host_rerror_rate: % of connections that have REJ errors from a destination host
Probe	▪ dst_host_diff_srv_rate: % of connections to different services from a destination host ▪ rerror_rate: % of connections that have REJ errors ▪ srv_diff_host_rate: % of connections that have same service to different hosts ▪ logged in: binary decision (1 successfully logged in, 0 failed login) ▪ service: type of service used to connect (e.g. fingure, ftp, telnet, ssh, etc.)
DoS	▪ count: number of connections made to the same host system in a given interval of time ▪ num_compromised: number of compromised conditions ▪ wrong fragments: number of wrong fragments ▪ land: binary decision (1 if connection is from/to the same host/port; 0 otherwise) ▪ logged in: binary decision (1 successfully logged in, 0 failed login)
U2Su	▪ root_shell: binary decision (1 if root shell is obtained; 0 otherwise) ▪ dst_host_srv_serror_rate: % of connections to the same service that have SYN errors from a destination host ▪ num_file_creations: number of file creations ▪ serror_rate: % of connections that have SYN errors ▪ dst_host_same_src_port_rate: % of connections to same service ports from a destination host
R2L	▪ guest login: binary decision (1 if the login is guest, 0 otherwise) ▪ num_file_access: number of operations on access control files ▪ destination bytes: number of bytes received by the source host form the destination host ▪ num_failed_logins: number of failed login attempts ▪ logged in: binary decision (1 successfully logged in, 0 failed login)

4.2 Ranking Algorithm Using Linear Genetic Programming

The performance of each of the selected input feature subsets is measured by invoking a fitness function with the correspondingly reduced feature space and training set and evaluating the intrusion detection accuracy. Once the required number of iterations is completed, the evolved high ranked programs are analyzed for how many times each input appears in a way that contributes to the fitness of the programs that contain them. The best feature subset found is then

output as the recommended set of features to be used in the actual input for the classifier. In the feature selection problem the main interest is in the representation of the space of all possible subsets of the given input feature set. Each feature in the candidate feature set is considered as a binary gene and each individual consists of fixed-length binary string representing some subset of the given feature set. An individual of length d corresponds to a d-dimensional binary feature vector Y, where each bit represents the elimination or inclusion of the associated feature. Then, $y_i = 0$ represents elimination and $y_i = 1$ indicates inclusion of the i^{th} feature. Fitness F of an individual program p is calculated as the Mean Square Error (*MSE*) between the predicted output (O_{ij}^{pred}) and the desired output (O_{ij}^{des}) for all *n* training samples and *m* outputs [9,10].

$$F(p) = \frac{1}{n \cdot m} \sum_{i=1}^{n} \sum_{j=1}^{m} (O_{ij}^{pred} - O_{ij}^{des})^2 + \frac{w}{n} CE = MSE + w \cdot MCE \quad (3)$$

Classification Error (*CE*) is computed as the number of misclassifications. Mean Classification Error (*MCE*) is added to the fitness function while its contribution is proscribed by an absolute value of Weight (W).

4.3 Ranking Algorithm Using Multivariate Adaptive Regression Splines

Generalized Cross Validation (GCV) is an estimate of the actual cross-validation which involves more computationally intensive goodness of fit measures. Along with the Multivariate Adaptive Regression Splines (MARS) [11] procedure, a generalized cross-validation procedure is used to determine the significant input features. Non-contributing input variables are thereby eliminated.

$$GCV = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{y_i - f(x_i)^2}{1 - k/N} \right] \quad (4)$$

where *N* is the number of records and *x* and *y* are independent and dependent variables respectively. *k* is the effective number of degrees of freedom whereby the GCV adds penalty for adding more input variables to the model. The contribution of the input variables may be ranked using the GCV with/without an input feature [12].

Table 5. Most Important Features Description as Ranked by MARS

Ranking Algorithm	Feature Description
Normal	▪ destination bytes: number of bytes received by the source host form the destination host ▪ source bytes: number of bytes sent from the host system to the destination system ▪ service: type of service used to connect (e.g. fingure, ftp, telnet, ssh, etc.) ▪ logged in: binary decision (1 successfully logged in, 0 failed login) ▪ hot: number of "hot" indicators
Probe	▪ dst_host_diff_srv_rate: % of connections to different services from a destination host ▪ dst_host_srv_count: : number of connections from the same host with same service to the destination host during a specified time window ▪ source bytes: number of bytes sent from the host system to the destination system ▪ dst_host_same_srv_rate: % of connections to same service ports from a destination host ▪ srv_count: : number of connections to the same service as the current connection during a specified time window
DoS	▪ count: number of connections made to the same host system in a given interval of time ▪ srv_count: : number of connections to the same service as the current connection during a specified time window ▪ dst_host_srv_diff_host_rate: % of connections to the same service from different hosts to a destination host ▪ source bytes: number of bytes sent from the host system to the destination system ▪ destination bytes: number of bytes received by the source host form the destination host
U2Su	▪ dst_host_srv_count: : number of connections from the same host with same service to the destination host during a specified time window ▪ count: number of connections made to the same host system in a given ▪ duration: length of the connection ▪ srv_count: : number of connections to the same service as the current connection during a specified time window ▪ dst_host_count: : number of connections from the same host to the destination host during a specified time window
R2L	▪ srv_count: : number of connections to the same service as the current connection during a specified time window ▪ count: number of connections made to the same host system in a given ▪ service: type of service used to connect (e.g. fingure, ftp, telnet, ssh, etc.) ▪ dst_host_srv_count: : number of connections from the same host with same service to the destination host during a specified time window ▪ logged in: binary decision (1 successfully logged in, 0 failed login)

5. Attacks on Intrusion Detection Systems

IDS play a vital role in a security chain, by alerting site administrators with all attempts to breach information security policy of an organization. In order for the IDS to be more useful in an information security chain, information system policy administrators needs to be able to rely on the information provided by the IDS; flawed systems not only provide false information about the current security scenario but also generate large volumes of false alarms. Moreover, the value of information from faulty systems is not only negated, but potentially misleading [13].

Most of the IDSs rely on several components (Data collection module, analysis module, storage module, and response module) to decide whether a particular activity is normal or malicious. Given the implications of malfunction of an IDS component, it is reasonable to assume that IDS are themselves logical targets for attack. Most of the time information security technologies become a primary target of a knowledgeable adversary. A potential adversary target's IDS components and can make the IDS absolute by disabling it or forcing it to provide false information (false alarms).

5.1. Vulnerabilities in Intrusion Detection Systems

All the components of an IDS are vulnerable to multiple attacks and have unique security implications on the functionality of IDS.

- Data collection module collects audit trails (user logs, network trails, system calls, etc.) for the other IDS components to decide whether a particular activity is malicious or normal. If an adversary attacks this module the IDS becomes un-functional.

- Analysis module analyses inputs (audit trails) form the data collection module to decide whether a particular activity is normal or malicious. If an adversary knows the analysis technique he can mislead and circumvent the IDS form being functional.

- Storage module provides a mechanism to store data collected by data collection and analysis modules in a secure fashion. Data stored might be used for building new signatures, updating user and system profiles, forensics analysis, and identifying key audit information. An adversary that can compromise the storage module can prevent the IDS from logging the attack information, insertion or deletion of audit trails. A more advanced adversary can also change the profiles and intrusion detection signatures of the IDS.

- Response module provides a mechanism for aftermath operations. A compromise on a response module will allow the adversary to continuously attack the system with out generating an alarm. In case of reactive devices rather than an aftermath devices an adversary can make the system deny legitimate activity and accept malicious activity.

5.2 Insertion and Evasion Attacks

Insertion attacks are caused by inserting malfunction packets that an end-system rejects but an IDS accepts [13]. An adversary exploits this feature, by sending packets that an end-system will reject but an IDS will still accept and inspect for malicious activity. Attacks range from insertion

of malfunction packets to data modification. Evasion attacks are caused by inserting legitimate packets that an IDS rejects but an end-system accepts. An adversary exploits this feature by sending packets that an end-system will accept but an IDS will reject. This will cause an IDS to generate false alarms and deny legitimate packets.

- **Bad Header Fields:** End-systems reject packets that have invalid header fields. Network peripheral devices do not route the packets with invalid header fields, but if the IDS is on the same local network as the adversary's its still subject to insertion attacks. Most of the IP implementations do not process packets with a bad checksum. An IDS that does check not check the packets for correct checksums is thus vulnerable to simple insertion attacks. Every packet requires a time to live (TTL) field to be routed by the routers. The router decrements the TTL value as it routes the packet to the next hop. An adversary can exploit this property by specifying a time that is just enough for the packets to reach the IDS but not the end-system.
- **IP Options:** Lack of proper knowledge of end-system implementations might lead an IDS to take ambiguous actions. Parsing of IP options varies from system to system; an IDS requires special processing capabilities for proper and correct interpretations. Most of the end-systems drop the packets if the IP checksum is wrong. An IDS with out the knowledge of the end-systems actions on wrong IP checksums, might lead it into an ambiguity, in taking a proper action.
- **Media access control (MAC) Address Spoofing:** Most of the network peripheral devices do not check for matching IP address and MAC address. Due to the use of complex dynamic protocols like dynamic hyper control protocol (DHCP), virtual private network (VPNs), network address translators (NAT), etc, it becomes even harder to verify the legitimacy. An adversary who knows the MAC address of the IDS, sends packet to it. An IDS that does not have the capability to check for legitimate IP address and MAC address pairs is subject to simple insertion attacks.

5.3. Availability Attacks

Recent trend of the adversaries "if I can't have it, nobody can" has changed the emphasis of information assurance with respect to information security. Legitimate networks packets consume various kinds of shared resources, such as bandwidth, memory, processing and operating system structures. Most of the IDS components require system and network resources to process the information passing by the network. An adversary identifies a few activities that are resource intensive and targets the IDS with such activity making it un-functional. A few possible scenarios of resource exhaustion are buffers exhausted, file descriptors exhausted, address space exhausted, disk space exhausted, CPU cycle exhausted, and banc with exhausted.

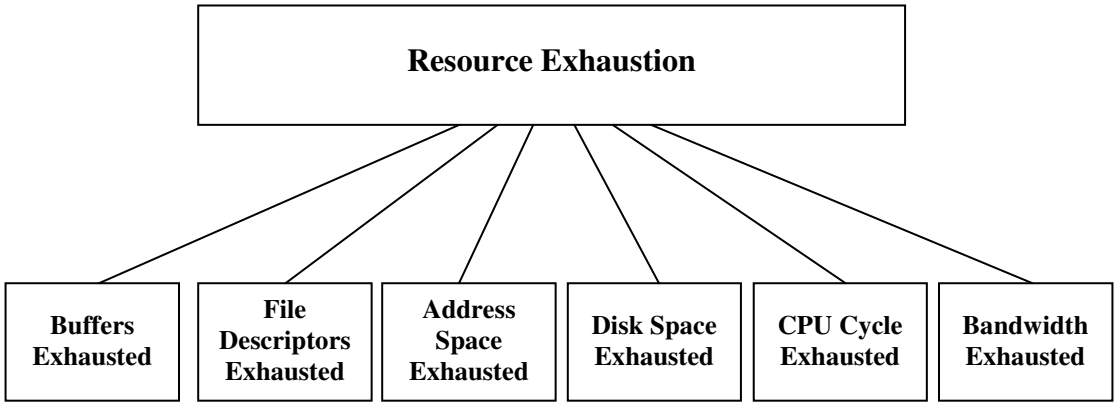

Figure 4. Resource Exhaustion Scenarios of an IDS

6. Attacks on Antivirus Tools

Software security assurance and malware detection are important aspects of information system assurance. Software obfuscation is a general technique that is used to protect the software from reverse engineering techniques and is being used by malware writers to circumvent the current detection mechanisms (Antivirus tools). Current static scanning techniques for malware detection have serious limitations; on the other hand, sandbox testing does not provide a complete solution either due to time constraints (e.g., time bombs cannot be detected before its preset time expires). In this section, we describe a few obfuscation techniques applied to recent viruses that were used to thwart commercial grade antivirus tools.

Malware Used for Analysis
Several recent viruses (executables) are being used for analysis. We describe the analysis of 4 viruses. The description of the virus is given based on the payload, enabling vulnerability, propagation medium and the systems infected.

i. **W32.Mydoom**: A mass mailing worm and a blended back door that arrives with as an attachment with file extensions .bat, .cmd, .exe, .pif, .scr or .zip [14]. The pay load performs a denial of service against www.sco.com and creates a proxy server for remote access using TCP ports 3127 through 3198. Infects all Windows systems.

ii. **W32.Blaster**: Exploits windows DCOM RPC vulnerability using TCP port 135. The pay load launches a denial of service attack against windowsupdate.com, might systems to crash and opens a hidden remote cmd.exe shell. Propagates via TCP ports 135, 4444 and UDP port 69. Infects only Windows 2000 and XP.

iii. **W32.Beagle**: A mass mailing worm blended with a back door. The worm contains large scale email with extensions, .wab, .htm, .xml, .nch, .mmf, .cfg, .asp, and etc [14]. Uses its own SMTP engine, TCP port 2745 to spread and also tries to spread via file sharing networks like Kazza. Infects all Windows systems.

iv. **Win32.Bika**: According to virus library it is a harmless per-process memory resident parasitic Win32 virus. It infects only Win32 applications [15]. The virus writes itself to the end of the file while infecting. Once the host program is infected it starts the virus hooks “set

current directory" Win32 API functions (SetCurrentDirectoryA, SetCurrentDirectoryW) that are imported by the host program and stays as a background thread of infected process, and then infects files in the directories when current directory is being changed. The virus does not manifest itself.

6.1 Obfuscation

In its simplest form obfuscation is obscuring some information such that another person cannot construe its true meaning. This is certainly true for code obfuscation where the objective is to hide the underlying logic of a program.

Code obfuscation has been compared to code optimization where code optimization is some transformation that will minimize a program's metric such as execution time or execution size while code obfuscation has the additional requirement that the code transformation also maximizes obscurity [16]. When we optimize for speed we generally try to take advantage of hardware pipelines, memory buffers, and such on while leaving the program essentially the same. Any optimization that changes the program's functionality or logic cannot be applied blindly and is generally avoided.

Obfuscation has also been applied to program watermarking and is a well known technique to prevent reverse engineering [17]. In general, obfuscating a program to prevent reverse engineering is similar to a classic cryptography game: you try and make reversing your obfuscation hard enough such that it is impractical to attack. Given enough time and resources any obfuscation can be reversed but as long as it takes 100,000 years it is considered pretty secure. By obfuscating you can prevent another individual from gaining knowledge about your program. With respect to malware, code obfuscation is an appealing technique to hinder detection. A simple obfuscation technique may render a known virus completely invisible to conventional scanners with very little effort on the part of the virus writer.

Applying an obfuscation transformation to a program has the advantage that it is essentially self-decrypting encryption. The code is rendered incomprehensible while still remaining a viable program.

Data obfuscation: changes the look of a program by modifying the constants or encapsulated bits of data. An example would be to split a string *hello world* into smaller strings, such as *he*, *ll*, and *o*. Another method would be to separate a Boolean variable into two integers and use comparisons between the two to emulate the True / False properties of the original.

In general, complex data transformations require the addition of "helper code" if the original functionality is to be maintained. In the example above, we would need to generate code to concatenate the small strings together to get the original "hello world" before using it.

Control flow obfuscation: control flow transformations focus on obscuring how the program runs. For example, inserting junk code into a program changes its appearance considerably but does not change the logic. A more complex example would be to use global pointers for control flow. If we used pointers *p* and *q*, and inserted statement like *if (p == q) then* it is nearly impossible to determine if this statement is true or false using static analysis. Such a

combination of pointers and control flow statements is considered opaque because of the difficulty inherent in pointer alias analysis.

This type of obfuscation is particularly appealing to malware authors because of its strength. We see control flow transformations implemented in polymorphic and metamorphic engines where the code changes with each host infected.

Other techniques: data and control flow are not the only techniques that can be used to obscure a program's meaning or prevent reverse engineering. Many software authors make use of *antidisassembly* and *antidebugging* techniques to hinder analysis. In general, these are "tricks" that slow down automated tools such as disassembles. Byte code scramblers are also used to obfuscate strongly typed bytecode such as Java's. All of these techniques, combined with a generous helping of data and control flow obfuscation, help make code analysis exorbitantly difficult.

6.1.1 Classification

For simplicity we have separated the obfuscation techniques into six general categories. Because of the complexity in implementing and detecting pointer aliases we gave them their own category. As a general rule the complexity and robustness of the technique increases the greater the type. Straight control flow obfuscation is (in general) not as robust as both data and control flow obfuscation together. These types assume a low level language such as x86 assembly.

Type 0: None

Program is left unmodified and functions exactly the same as before.

Type 1: Null operations

NOPs are inserted into the code. There is virtually no modification to data or control flow. An example of a type 1 transformation is presented in Table 6 below. On the left we have the original code and on the right we have the modified code with null operations inserted every 2nd operation.

Table 6. Null operations obfuscation

Original Code			After Transformation	
mov	eax, -44(ebp)		mov	eax, -44(ebp)
mov	-44(ebp), ebx		mov	-44(ebp), ebx
sub	12, esp		**nop**	← Null operation
lea	-24(ebp)		sub	12, esp
push	eax		lea	-24(ebp)
			nop	← Null operation
			push	Eax

Inserting null operations is essentially the same as inserting white space in a document: it may take longer to read but the content is exactly the same.

Type 2: Data

Some data obfuscation transformation is applied, such as string splitting or variable type replacement. For example, we could replace a Boolean variable with two integers. If they are

equal, the statement is true, otherwise it is false. In the example below Table 7, ***x*** is a Boolean variable and ***a*** and ***b*** are integers. The code on the left is the original control flow and the code on the right performs exactly the same but has a different signature.

Table 7. Null operations obfuscation

Original code and meaning				Transformed code and meaning		
`cmpb`	0, x	if (x == true)		mov	a, eax	if (a < b)
`je`	.sub	goto sub		cmpl	b, eax	goto sub
				jge	.sub	

Type 3: Control flow

Control flow transformations are applied. Code is swapped around and jump instructions are inserted. For example, we could copy the contents of a subroutine to another location in the file and add jumps to and from the subroutine. The code would function exactly the same but look quite different. In Table 8 below, three lines of code have been shifted to some location (denoted as **[shift]**) and helper code has been inserted.

Table 8. Control flow obfuscation

Original code			After transformation		
cmp	24, eax		**jmp**	**[shift]**	
jne	.sub		**nop**		← Helper code
sub	12, eax		**nop**		
push	eax		push	eax	← Original execution path resumes
	…			…	
			cmp	24, eax	
			jne	.sub **– [shift]**	
			sub	12, eax	
			jmp	**-[shift]**	← Helper code

Table 9. Combination of null operations and control flow obfuscation

Original code			After transformation		
cmp	24, eax		**jmp**	**[shift]**	
jne	.sub		**nop**		← Helper code
sub	12, eax		**nop**		
push	eax		push	eax	← Original execution path resumes
	…			…	
			mov	24, eax	← Data obfuscation
			cmpl	b, eax	
			jle	.dead_code	
			jne	.sub **– [shift]**	
			sub	12, eax	
			jmp	**-[shift]**	← Helper code

Type 4: Combination of 2 and 3
We pull out all the stops and combine data and control flow transformations. At this level junk code is inserted and variables can be completely replaced with large sections of needless code. For example, we can modify all integer variables as above and transpose the program's entry point as in Table 9.

Type 5: Pointer aliasing
The final step is to introduce pointer aliasing. Variables are replaced with global pointers and functions are referred to by arrays of function pointers. This type of transformation is relatively easy to implement using high level languages that allow pointer references but tricky (at best) using assembly languages. Pointer aliasing can be as simple as changing ***a = b*** into ****a = **b*** or as complex as converting all variables and functions into an array of pointers referenced by pointers to pointers.

6.1 Obfuscation Used for Defeating Commercial Scanners
In our research, we discovered that most commercial virus scanners could be defeated with very simple obfuscation techniques. For example, simple program entry point modifications consisting of two extra jump instructions effectively defeated most scanners. Therefore, we only used the bare minimum level of obfuscation needed to prevent detection. Our goal was to show how trivial it is to modify recent malware to defeat existing scanning techniques using only the compiled executable and a few tools.

The obfuscation process is presented in Figure 4. The binary code is disassembled into a more readable format so that we may understand what the program is doing. Someone with foreknowledge about the malware need not spend so much time analyzing the program. Once we have the disassembled program and have had given it study we pick out an area to attack. The first target when applying a control flow transformation is to attack the program's entry point but when using a data transformation we generally have to take a guess. We decide where and what modifications need to be performed and change the binary file directly, using the disassembled version as a guide or map. Once all modifications have been made, the file is examined using the anti-virus scanners.

All variants with the exception of the *MyDoom* virus were generated using of the shelf hex editing tools. We were fortunate enough to have a copy of the *MyDoom.A* source code and made all our modifications using the *Microsoft Visual* development suite. The *Hackman* hex editing utility was used to generate all other variants [18].

Table 10 shows the preliminary results of New Mexico Tech's recent investigation of the MyDoom worm and several other recent worms and viruses, using eight different (commercial) scanners and proxy services. (✓ indicates detection, ✖ indicates failure to detect, and ? indicates only an "alert"; all scanners used are most current and updated version) [19,20].

The obfuscation techniques used to produce the polymorphic versions of different malware tested in the experiments include control flow modification (e.g. Mydoom V2, Beagle V2), data segment modification (e.g., Mydoom V1, Beagle V1), and insertion of dead code (e.g., Bika V1).

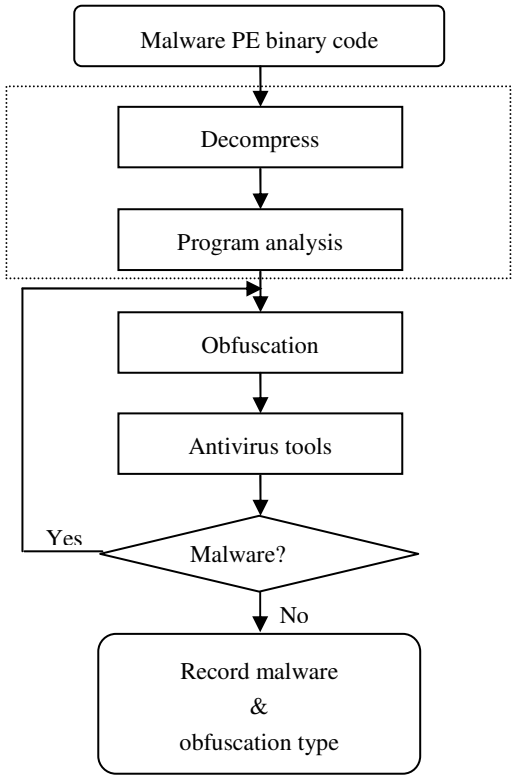

Figure 4. Obfuscation Attack Process on Commercial Scanners

Table 10. Obfuscation Attacks on Commercial Scanners

	N	M^1	M^2	D	P	K	F	A
W32.Mydoom.A	✓	✓	✓	✓	✓	✓	✓	✓
W32.Mydoom.A V1	×	✓	✓	×	×	✓	✓	×
W32.Mydoom.A V2	✓	×	×	×	×	×	×	×
W32.Mydoom.A V3	×	×	×	×	×	×	×	×
W32.Mydoom.A V4	×	×	×	×	×	×	×	×
W32.Mydoom.A V5	×	?	×	×	×	×	×	×
W32.Mydoom.A V6	×	×	×	×	×	×	×	×
W32.Mydoom.A V7	×	×	×	×	×	×	×	×
W32.Bika	✓	✓	✓	✓	✓	✓	✓	✓
W32.Bika V1	×	×	×	✓	×	✓	✓	✓
W32.Bika V2	×	×	×	✓	×	✓	✓	✓
W32.Bika V3	×	×	×	✓	×	✓	✓	✓
W32.Beagle.B	✓	✓	✓	✓	✓	✓	✓	✓
W32.Beagle.B V1	✓	✓	✓	×	×	✓	✓	×
W32.Beagle.B V2	✓	×	×	×	×	×	×	×
W32. Blaster.Worm	✓	✓	✓	✓	✓	✓	✓	✓
W32. Blaster.Worm V1	×	✓	✓	✓	✓	✓	✓	×
W32. Blaster.Worm V2	✓	✓	✓	×	×	✓	✓	×
W32. Blaster.Worm V3	✓	✓	✓	✓	✓	×	×	×
W32. Blaster.Worm V4	×	×	×	×	×	✓	✓	×

N – Norton**, M^1 –** McAfee UNIX Scanner**, M^2 –** McAfee**, D –** Dr. Web**, P –** Panda**, K –** Kaspersky**, F –** F-Secure**, A –** Anti Ghostbusters

7.0. Conclusions

In this chapter we attempted to present the current cyber security challenges from an intrusion detection system and antivirus tools perspective. The state-of-the-art of the evolution of intrusion

detection technology with an overview of computer attack taxonomy and computer attack demystification along with a few detection signatures is presented.

Because malware is expected to become more lethal ("3rd generation" worms using multiple attack vectors to exploit both known and unknown vulnerabilities) and spreads even faster (attacking pre-scanned targets with lightning speed) in the future, it is important that the scanners are capable of detecting polymorphic (obfuscated, or variant) versions of known malware. The currently available scanners, however, are grossly inadequate since they are not able to detect even slightly obfuscated versions of known malware, as shown in Table 3.

Acknowledgements

Authors would like to thank Professor Rao Vemuri for the editorial comments which improved the presentation of this paper.

References

1. Kumar S., Spafford E. H. (1994) "An Application of Pattern Matching in Intrusion Detection," Technical Report CSD-TR-94-013. Purdue University
2. Denning D. (1987) "An Intrusion-Detection Model," IEEE Transactions on Software Engineering, SE-13 (2): 222-232.
3. Kendall K. (1998) "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems", *Master's Thesis, Massachusetts Institute of Technology*.
4. Webster S. E. (1998) "The Development and Analysis of Intrusion Detection Algorithms," *M.S. Thesis, Massachusetts Institute of Technology*.
5. Mukkamala S., and Sung. A. H. (2003) Feature Selection for Intrusion Detection Using Neural Networks and Support Vector Machines. Journal of the Transportation Research Board of the National Academics, Transportation Research Record No 1822; 1822: 33-39.
6. Mukkamala S., and Sung. A. H. (2003) Identifying Significant Features for Network Forensic Analysis Using Artificial Intelligence Techniques. In International Journal on Digital Evidence, IJDE; 1.
7. Vladimir V. N. (1995) The Nature of Statistical Learning Theory. Springer.
8. Joachims T. (2000) Making Large-Scale SVM Learning Practical. LS8-Report, University of Dortmund.
9. Banzhaf. W, Nordin P., Keller E. R., and Francone F. D. (1998) Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs and its Applications. Morgan Kaufmann Publishers, Inc.
10. Brameier. M, Banzhaf. W. (2001) A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining. IEEE Transactions on Evolutionary Computation; 5(1): 17-26.
11. Friedman J. H. (1991) Multivariate Adaptive Regression Splines. Annals of Statistics; 19: 1-141.

12. Steinberg D., Colla. P. L., and Kerry. (1999) MARS User Guide. Salford Systems, San Diego.
13. Ptacek H. T., and Newsham N. T. (1998) Insertion, Evasion and Denial of Service: Eluding Network Intrusion Detection. Secure Networks Inc.
14. Symantec Cooperation http://securityresponse.symantec.com/avcenter/ (accessed on September 16, 2004)
15. Virus Library http://www.viruslibrary.com/virusinfo/Win32.Bika.htm (accessed on September 16, 2004)
16. Collberg and Thomborson C. (2002) Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection, IEEE Transactions on Software Engineering 28:8, 735-746
17. Krishnaswamy S., Kwon M., Ma D., Shao Q., and Zhang Y. (2000) Experience with software watermarking, In the Proceedings of 16th Annual Computer Security Applications Conference, pp. 308-316.
18. Hex H. (Editor), http://www.technologismiki.com/en/index-h.html.
19. Sung A. H., Xu J., Ramamurthy. K, Chavez P., Mukkamala S., Sulaiman T., Xie T. (2004) Static Analyzer for Vicious Executables (SAVE). *Presented in Work-in-progress Section of IEEE Symposium on Security and Privacy*.
20. Sung A. H., Xu J., Chavez P., Mukkamala S. (2004) Static Analyzer for Vicious Executables (SAVE). In the Proceedings of 20th Annual Computer Security Applications Conference (To appear).