PROJECT REPORT ON

Intrusion Detection Systems

BY

DHAWAL KHEM (06305021)

HARIN VADODARIA (06305015)

MANISH AGGARWAL (06305005)

MITESH M. KHAPRA (06305016)

NIRAV UCHAT (06305906)

UNDER THE GUIDANCE OF

PROF. BERNARD L. MENEZES

K.R.SCHOOL OF INFORMATION TECHNOLOGY,

INDIAN INSTITUTE OF TECHNOLOGY, BOMBAY

MUMBAI

Contents

ABSTRACT

The purpose of this report is to introduce the user to Intrusion Detect Systems and give a deep understanding of some sophisticated techniques for intrusion detection. Intrusion Detection is an important component of infrastructure protection mechanisms. Given the increasing complexities of today's network environments, more and more hosts are becoming vulnerable to attacks and hence it is important to look at systematic, efficient and automated approaches for Intrusion Detection. Here we discuss some data mining based approaches for intrusion detection and compare their merits and demerits. We also look at some signature based detection techniques for detecting polymorphic worms. We also look at various port scanning techniques and discuss some techniques for detecting port scanning attempts. We then discuss the architecture of an advance Intrusion Detection System, Snort and suggest some enhancements to the same.

Keywords: Intrusion Detection, Data Mining, Polymorphic worms, Signature based detection, Anomaly based detection, Snort, Port Scanning.

Chapter 1: What is an IDS?

1.1. Introduction

An Intrusion Detection System is used to detect all types of malicious network traffic and computer usage that can't be detected by a conventional firewall. This includes network attacks against vulnerable services, data driven attacks on applications, host based attacks such as privilege escalation, unauthorized logins and access to sensitive files, and malware (viruses, trojan horses, and worms).

An IDS is composed of the following three components:

Sensors: - which sense the network traffic or system activity and generate events.

Console: - to monitor events and alerts and control the sensors,

Detection Engine: - that records events logged by the sensors in a database and uses a system of rules to generate alerts from the received security events.

There are several ways to categorize an IDS depending on the type and location of the sensors and the methodology used by the engine to generate alerts. In many simple IDS implementations all three components are combined in a single device or appliance.

1.2. Types of Intrusion-Detection systems

Network Intrusion Detection System: - identifies intrusions by examining network traffic and monitors multiple hosts. Network Intrusion Detection Systems gain access to network traffic by connecting to a hub, network switch configured for port mirroring, or network tap. An example of a NIDS is Snort.

Host-based Intrusion Detection System: - consists of an agent on a host which identifies intrusions by analyzing system calls, application logs, file-system modifications (binaries, password files, capability/acl databases) and other host activities and state.

Hybrid Intrusion Detection System: - combines one or more approaches. Host agent data is combined with network information to form a comprehensive view of the network. An example of a Hybrid IDS is Prelude.

1.3. Passive system v/s reactive system

In a **passive system**, the IDS sensor detects a potential security breach, logs the information and signals an alert on the console. In a reactive system, which is known as an Intrusion Prevention System (IPS) the IDS responds to the suspicious activity by resetting the connection it believes to be suspicious or by reprogramming the firewall to block network traffic from the suspected malicious source, either autonomously or at the command of an operator.

Though they both relate to network security, an IDS differs from a firewall in that a firewall looks outwardly for intrusions in order to stop them from happening. The firewall limits the access between networks in order to prevent intrusion and does not signal an attack from inside the network. An IDS evaluates a suspected intrusion once it has taken place and signals an alarm. An IDS also watches for attacks that originate from within a system.

1.4. Signature Based Detection v/s Anomaly Based Detection

Signature based detection:-

This detection technique uses specifically known patterns to detect malicious code. These specific patterns are called signatures. Identifying the worms in the network is an example of signature based detection.

Anomaly Detection:-

These techniques are designed to detect abnormal behavior in the system. The normal usage pattern is baselined and alerts are generated when usage deviates from the normal behavior.

Example if a user logs on and off 20 times a day while the normal behavior is 1-2 times.

Chapter 2: Signature Based Detection of Worms and Polymorphic Worms

2.1. Worms v/s Viruses

A worm is any malicious code that has the capability to replicate and spread on its own. It works on the scan, compromise and replicate principle. First it scans the network to find hosts having vulnerabilities and then exploits these vulnerabilities to compromise the target and finally replicates itself on the target. Viruses, on the other hand, can't spread on their own. They attach to some other programs and depend on these programs to spread in the network.

Every worm has a unique bit string which can be used to identify the worm (i.e. all instances of the worm in the network have the same bit string representation).

2.2. Detecting worms – A simple technique

1. Identify the worms using honey pots.
2. Manually extract the signature.
3. Make the signature public, so that IDS or any other anti virus software can update their signature database.
4. Now the IDS can check each incoming or outgoing packet and compare it with the stored signature and raise an alert if a match is found.

This technique is not very effective because of the following reasons.

- *Speed with which worm spreads:* Worms can spread at enormous speeds. E.g.
  The Sapphire/slammer worm infected more than 75,000 vulnerable hosts in less than 10 minutes.
  Hence any technique which involves manual extraction of worms will fail to match the speed at which worms spread. By the time signature of the worm is identified, millions of hosts would have been infected.
- *Zero day worms:* The above technique will fail against zero day worms. Zero day worms are those worms that exploit the vulnerabilities that have not been declared yet or the worms that start spreading as soon as (on the same day) some vulnerability is made public.

This encouraged us to study some techniques for automatic extraction of signatures. We discuss some of these techniques here.

2.3. Content Sifting Approach [1]

This approach works on the following assumptions:-

- Some part of the worm representation is invariant i.e. there will always be some portion of the worm's body that remains same across every instance of the worm.
- Spreading of worms in the network is very different from the normal network traffic. For example when a worm spreads we see the same byte string in different packets exchanged between many sources and destinations. This pattern is very unlikely in normal network traffic and applications.

2.3.1. Steps to extract the signature

1. Look for the packets in the network that have common sub strings of sufficiently large length (say x) and are exchanged between many systems in the network.
2. Maintain a table to store an entry for each substring of length x appearing in any network packets.
3. Keep track of the number of times we see each such sub string in the network, and also the distinct source IP and destination IP address corresponding to that substring.
4. If all the three values (i.e. number of times the string appeared, the number of sources from which it originated and the number of destinations to which it was sent) cross some predefined threshold value we declare it as a worm and that substring will be stored as the signature of the worm

2.3.2. A critic of the Content Sifting approach

The above technique will fail if the worms somehow change their representation before spreading so that no two instances of the worm have the same representation and hence no two instances of the worm will have any substring (of sufficient length) in common.

2.4. Polymorphic Worms (PW)

Every worm has a unique bit string which can be used to identify the worm (i.e. all instances of the worm in the network have the same bit string representation). Hence worms can be detected easily using simple signature based techniques (i.e. by comparing the network packets against a database of known signatures). Polymorphic worms, on the other hand, change their representation before spreading i.e. each instance of a polymorphic worm will have a different bit stream representation.

2.4.1. How does a worm achieve polymorphism?

Encryption

Here, the worm encrypts its body with a random key each time before spreading. A small executable code is then attached to the body of the worm. This executable code is responsible for decrypting the encrypted body of the worm on the victim's machine and then gives control to the worm.

Code substitution

Here, the instructions in the worm body are substituted with semantically equivalent instructions. Some examples are mentioned below.

1. Multiplication can be achieved by successive addition.
2. Addition can be achieved using xor operator.
3. Register renaming: if you want to transfer a value from register B to A, first move the value to any unused register and then move it to A.

By doing this, although the behavior of the program remains the same the representation of the byte string changes.

2.4.2. Parts of a polymorphic worm

Body/Code of the worm

This is the part of the worm which is malicious and does the actual damage.

Polymorphic Engine (PE)

The polymorphic engine is responsible for changing the representation of the worm either by encryption, code substitution or both.

Polymorphic Decryptor (PD)

The polymorphic decryptor is responsible for decrypting the worm (if encryption technique is used for polymorphism) on the victim's machine and then give control to the worm.

Please note that irrespective of the technique used to achieve polymorphism the worm will always have some part which is executable. We should try to exploit this property of polymorphic worms and try to extract their signature based on this executable part.

2.4.3. Detecting Polymorphic worms

As discussed earlier, the content sifting approach used for normal worms will fail for polymorphic worms. This motivated us to look at some other techniques for detecting

polymorphic worms. These techniques focus on the idea that the signature of a worm should not be a function of the plain byte string representation of the worm body but it should be a function of some unique property of the worm that does not change with each new instance of the worm. A key observation is that the internal structure of the executable in the worm is more characteristic than its representation as a stream of bytes. So if we can map the structural behavior of the executable appearing in the worm body to a unique number, we can use that unique number as the signature of the newly discovered worm. Let us now discuss one technique which actually does this.

2.5. Control Flow Graph based approach for detecting Polymorphic Worms [2]

The following steps are involved in this approach:-

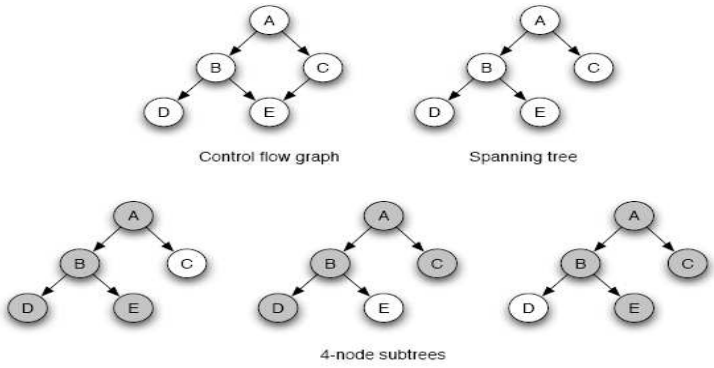
2.5.1. Construction of the Control Flow Graph (CFG)

First of all we make a control flow graph for each network packet. For this, we perform a linear disassembly of the byte stream by using any general purpose disassembler. Based on the instruction sequence, create the control flow graph. A CFG consists of nodes (Basic Blocks) and edges. A basic block is a sequence of instructions without any jumps or jump target in the middle of the block. We draw an edge from one node to another if there is a jump from the corresponding basic block to another.

Here, the question arises that how do we find the executable part in the network packets? The answer is that there is no need to find the executable part. This is because of the fact that when we build a CFG of a binary code, it will contain a large cluster of closely connected nodes whereas the CFG of a random sequence of bytes will contain small clusters or isolated nodes. So we can ignore all isolated nodes and small clusters.

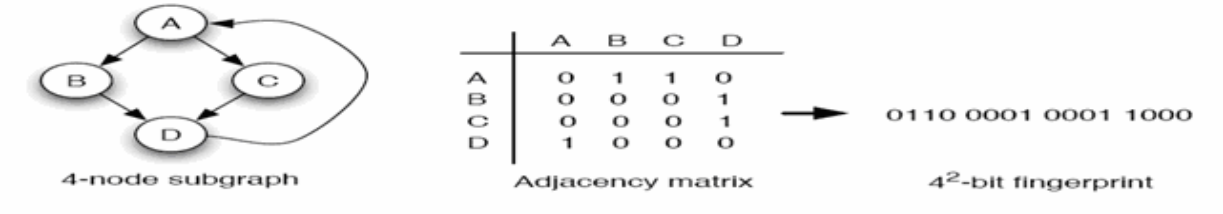
2.5.2. Graph Fingerprinting

Once we have constructed the control flow graph we need to extract a unique signature from this control flow graph. For this we construct all k-node sub graphs of the CFG. The value of k is decided heuristically by observing known worms (E.g. Slammer worm was the smallest worm to be detected till date and its CFG had 14 nodes). We first create a spanning tree to reduce redundant sub graphs and then find these k-node sub graphs.



Extracting 4-node sub graphs. (Image from [2])

We now want to map each of these k-node sub graphs to some unique number. For this, we convert each sub graph into its canonical form. Canonical form of a graph is a form in which all the isomorphs of the graph have the same representation. Now, create an adjacency matrix of each sub graph and then concatenate each row of the adjacency matrix to get the fingerprint of the sub graph.



Creating Fingerprint of a sub graph. (Image from [2])

2.5.3. Graph coloring

Till now we haven't given any importance to the instructions that are appearing in the basic blocks. So there may be a rare chance of getting the same control flow graph for two different binaries. To prevent this, the concept of graph coloring is introduced.

Class	Description	Class	Description
Data Transfer	`mov` instructions	String	x86 string operations
Arithmetic	incl. shift and rotate	Flags	access of x86 flag register
Logic	incl. bit/byte operations	LEA	load effective address
Test	test and compare	Float	floating point operations
Stack	push and pop	Syscall	interrupt and system call
Branch	conditional control flow	Jump	unconditional control flow
Call	function invocation	Halt	stop instruction execution

A possible distribution of instructions into classes [2]

Here, we divide the instructions into 14 sets or classes. A 14 bit color value is associated with each node, 1 bit corresponding to 1 class/set. Whenever one or more instructions of certain class appear in the basic block, the corresponding bit of the basic block color value is set to 1. Append this 14 bit color value to each node in the adjacency matrix of the sub graph. Concatenate the rows as before and get the new fingerprint. If the instructions are divided into classes carefully such that semantically equivalent instructions fall in the same class then it makes the instruction substitution techniques much less effective.

2.5.4. Identifying and detecting worms

Maintain a set of network stream s_i for each given fingerprint f_i. Every set s_i contains distinct source destination pairs of the stream that contains f_i. If the following conditions are met then we can say that a worm has been identified:

- The distinct source destination IP address pairs for a given f_i should be greater than some threshold value (say M).
- The number of distinct internal hosts appearing in s_i should be at least 2.
- The number of distinct external hosts should be at least 2.

2.5.5. A critic of the CFG based approach

- As the procedure is complex and time consuming it is difficult to handle high speed network stream.
- If an intelligent hacker comes up with a code substitution technique which completely alters the CFG then this technique will fail.
- This technique will fail the worm body doesn't contain any executable part (and there are some research scientists who are looking at methods for developing polymorphic worms which do not contain any executable part!!).

Chapter 3: Anomaly Based Detection

3.1. Motivation

An Anomaly-Based Intrusion Detection System is a system for detecting computer intrusions and misuse by monitoring system activity and classifying it as either normal or anomalous. The classification is based on heuristics or rules, rather than patterns or signatures, and will detect any type of misuse that differs significantly from normal system operation. Earlier, IDSs relied on some hand coded rules designed by security experts and network administrators. However, given the requirements and the complexities of today's network environments, we need a systematic and automated IDS development process rather than the pure knowledge based and engineering approaches which rely only on intuition and experience. This encouraged us to study some Data Mining based frameworks for Intrusion Detection. These frameworks use data mining algorithms to compute activity patterns from system audit data and extract predictive features from the patterns. Machine learning algorithms are then applied to the audit records that are processed according to the feature definitions to generate intrusion detection rules.

The Data Mining based approaches that we have studied can be divided into two main categories:-

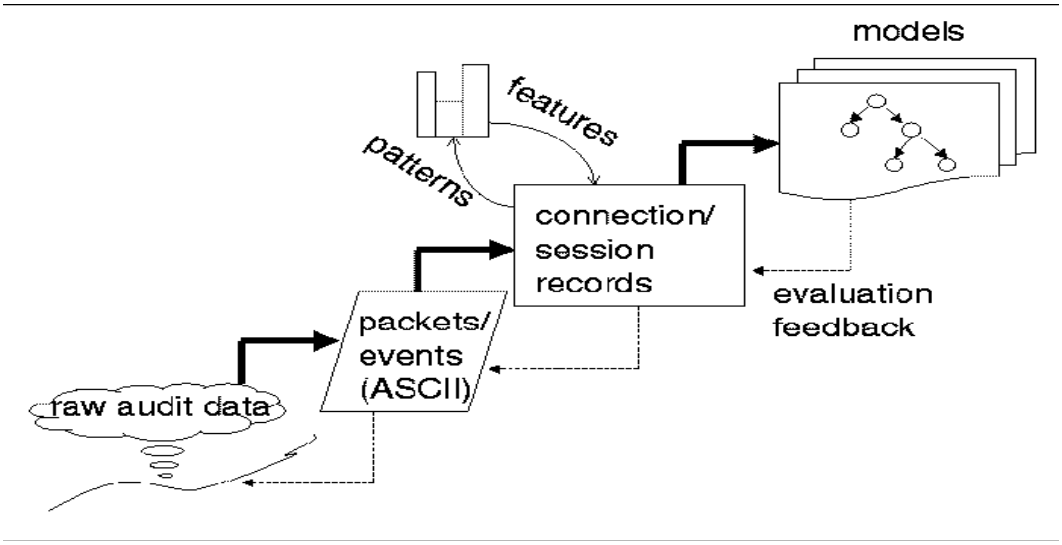
1. Supervised Learning
    a. Associative Rule Mining

2. Unsupervised Learning
    a. Clustering

We will discuss the above approaches and compare their merits and demerits.

3.2. Supervised Learning Based Approach → Associative Rule Mining (MADAM ID Framework) [3]

Here we will be discussing the MADAM ID (Mining Audit Data for Automated Models for Intrusion Detection) framework proposed in [3].The MADAM ID framework can be summarized as shown in the figure below:-



Data Mining process of building Intrusion Detection Systems
Image from (3)

Raw (binary) audit data is first processed into ASCII network packet information (or host event data), which is in turn summarized into connection records (or host session records) containing a number of basic features, such as service, duration, source IP address, destination IP address *etc.* Data mining programs are then applied to the connection records to compute the frequent patterns (i.e., association rules and frequent episodes), which are in turn analyzed to construct additional features for the connection records. Classification programs, for example, RIPPER [Cohen 1995], are then used to inductively learn the detection models.

Let us now look at the two main components of this framework viz.: -

1. Mining Audit Data.
2. Identifying Intrusion Patterns

3.2.1. Mining Audit Data

We mine the audit data to generate association rules that define the correlations between attributes. Given a set of records, where each record is a set of items, support(X) is defined as the percentage of records that contain item set X. An association rule is an expression of the form

$X \rightarrow Y, [c, s]$.

Where,

X, Y are item sets and $X \cap Y = \emptyset$

s = support (X U Y) = % of records containing X U Y

c = confidence = support (X U Y)/ support (X)

Consider the following set of shell commands executed by a secretary.

Time	Hostname	Command	arg1
am	pascal	vi	tex
am	pascal	tex	vi
am	pascal	mail	fredd
am	pascal	subject	progress
am	pascal	vi	tex
am	pascal	vi	tex
am	pascal	mail	williamf
am	pascal	subject	progress
...	...	...	...
am	pascal	vi	tex
am	pascal	latex	tex

Example taken from (3)

A data mining algorithm is applied to the above audit data to find the frequent patterns (i.e. to study the normal behavior of the secretary). The original association rules algorithm searches for all possible frequent associations among the set of given features. However, not all associations are necessarily useful for analyzing program or user behavior. Certain features are essential in describing the data, while others provide only auxiliary information. Domain knowledge is used to determine the appropriate essential features for an application. In shell command data, since the combination of the exact "time" and "command" uniquely identifies each record, "time" and "command" are the essential features; likewise, in network connection data, timestamp, source and destination hosts, source port, and service (i.e., destination port) are the essential features because their combination uniquely identifies a connection record. These essential features(s) are called axis features. The association rules algorithm is restricted to only output rules that include axis feature values.

Example, (3)

Association Rules from the Shell Command data	Meaning
command = vi → time = am,hostname = pascal, arg1 = tex, [1.0, 0.28]	When using vi to edit a file, the user is always i.e., 100% of the time) editing a tex file, in the morning, and at host pascal; and 28% of the command data matches this pattern.
command = subject → time = am,hostname = pascal, arg1 = progress, [1.0, 0.11]	The subject of the user's email is always (i.e.100% of the time) about "progress", such emails are in the morning, and at host pascal; and 11% of the command data matches this pattern.

3.2.2. Frequent Episodes

In the case of network connections we are often interested in studying the frequent sequential patterns rather than finding associations amongst attributes within a single record.

Frequent episodes are used to represent the sequential audit record patterns. Given a set of time stamped event records, where each record is a set of items, an interval [t1, t2] is the sequence of event records that starts from timestamp t1 and ends at t2. The width of the interval is defined as t2 - t1. Let X be a set of items; an interval is a minimal occurrence of X if it contains X and none of its proper subintervals contains X. A frequent episode rule is an expression of the form:-

$$X, Y \rightarrow Z \; [c, s, w]$$

where,

s = support = support ($X \cup Y \cup Z$)

c = confidence = support ($X \cup Y \cup Z$)/support ($X \cup Y$)

w = window = the time frame within which the events occur

Example (3), **(**Frequent Episode Pattern for SYN flood attack)

Frequent Episode	**Meaning**
flag = S0, service = http, dst_host = victim *flag = S0, service = http, dst_host = victim* *flag = S0, service = http, dst_host = victim*	93% of the time, after two http connections with S0 flag are made to host victim, within 2 seconds from the first of these two, the third similar connection is made, and this pattern occurs in 3% of the data

3.2.3. Identifying the Intrusion Patterns

The frequent episodes program is applied to both the exhaustively gathered normal connection dataset and the dataset that contains an intrusion. The resulting patterns are compared to find the intrusion-only patterns, that is, those that exhibit only in the intrusion dataset. The details of the pattern comparison algorithm are described in [6]. Briefly, the idea is to first convert patterns into numbers in such a way that "similar" patterns are mapped to "closer" numbers. Then pattern comparison and intrusion pattern identification are accomplished through comparing the numbers and rank ordering the results.

3.3. Unsupervised Learning Based Approach → Clustering [7]:

The approach that we discussed earlier requires a set of purely normal data from which a model is trained. However, more often than not we do not have either labeled or purely normal data readily available. Here, we present a new type of intrusion detection algorithm, unsupervised anomaly detection or clustering, to address the problem of lack of labeled data. This algorithm takes as input a set of unlabeled data and attempts to find intrusions within the data. This approach makes the following two assumptions about the data as mentioned below: -

1. The number of normal instances vastly outnumbers the number of intrusions.
2. Intrusions are very different from normal data.

Based on the above assumptions, the clustering approach clusters the data instances together into clusters using a simple distance-based metric. This clustering is performed on unlabeled data. Once the data is clustered, we label all those instances that appear in small clusters as anomalies (since intrusions are rare and small in number).

3.3.1. Algorithm

The algorithm proceeds as follows:-

1. Initialize the set of clusters, S, to the empty set.
2. Obtain a data distance d from the training set. If S is empty, then create a cluster with d as the defining instance, and add it to S. Otherwise, find the cluster in S that is closest to this instance. In other words, find a cluster C in S, such that for all C_i in S, dist(C, d) <= dist (C_i, d).
3. If dist (C,d) <= W, then associate d with the cluster C. Otherwise, d is more than W away from any cluster in S, and so a new cluster must be created for it: S → S U (C_m) where C_m is the cluster d as its defining distance.
4. Repeat steps 2 and 3, until no instances are left in the training set.

3.3.2. Labeling clusters

Once the clusters have been created by running the above algorithm we need to label these clusters as “normal” and “anomaly”. We do this based on the above mentioned assumptions.

1. Large clusters i.e. clusters having a large number of instance are classified as “normal”
2. Small clusters which are at a large distance form the bigger clusters are classified as “anomaly” (intrusions are rare and normal)

3.3.3. Detection

Given a new instance d, the classification proceeds as follows:-

1. Find a cluster which is closest to d under the distance metric M.
2. Classify d according to the label of C (either normal or anomaly)

3.4. Comparisons and critic

Approach	Average Detection Rate	Average False Positive Rate
MADAM ID	65-70 %	5%
Clustering	45-55%	1.3-2.3%

1. Supervised learning gives a better detection rate as compared to unsupervised learning with acceptable false positive rates.
2. Supervised learning requires huge amount of labeled data which requires a lot man power and is hence expensive.
3. Unsupervised learning performs better than supervised learning for new types of intrusions as on as these intrusions conform to the basic assumptions made by the clustering algorithm.

Chapter 4 Future Work

4.1. Polymorphic Worms

During our project work we read some literature which described some novel ways of constructing polymorphic worms which makes it difficult to detect these worms. We intend to study these methods and see if the CFG based approach works against such worms. We also intend to come up with some methods to generate polymorphic worms which completely alter the control flow graph of the executable code and then design some technique to detect such worms.

4.2. Data Mining Based Approaches for Anomaly Detection

We have implemented the Apriori algorithm [4] for associative rule mining. We intend to make some enhancements to this algorithm to improve its efficiency and reduce the training time by using some bitmap based optimization techniques. We also intend to develop a detection engine that does a real time detection of intrusions based on the rules generated by our mining algorithm. This detection will use a cost sensitive model as described in [8] so that the speed of detection increases. Instead of looking at all temporal statistics and features of the data this cost sensitive model will divide the features into four groups (based on their computational cost). The detection will first use the less expensive features to detect anomalies and proceed to the high cost features only if these low cost features fail to make an accurate prediction (based on some threshold). We also intend to develop a small module to convert the rules generated by our mining algorithm into Snort-compatible rules as described in [11].

Bibliography

[1] S. Singh, C. Estan, G. Varghese and S. Savage, “Automated worm fingerprinting”. In OSDI, 2004.

[2] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna,“Polymorphic Worm Detection Using Structural Information of Executables”. In 8th International Symposium on Recent Advances in Intrusion Detection (RAID), 2005

[3] Lee, W. and Stolfo, S. J., "A Framework for Constructing Features and Models for Intrusion Detection Systems," ACM Transactions on Information and System Security, vol. 3, November, 2000

[4] Lee, W., Stolfo, S., and Mok, K. (1999d), “Algorithms for Mining System Audit Data”, In Lin, T. Y. and Cercone, N., editors, Data Retrieval and Data Mining. Kluwer Academic Publishers.

[5] W. Lee, S. J. Stolfo, and K. W. Mok, “A data mining framework for building intrusion detection models”, In Proceedings of the 1999 IEEE Symposium on Security and Privacy, May 1999.

[6] W. Lee, S. J. Stolfo, and K. W. Mok, “Mining in a data-flow environment: Experience in network intrusion detection”, In Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-99).

[7] Leonid Portnoy, Eleazar Eskin, and Salvatore J. Stolfo., “Intrusion detection with unlabeled data using clustering”, In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001).

[8] W. Lee, S. Stolfo, P. Chan, E. Eskin, W. Fan, M. Miller, S. Hershkop, J. Zhang. Real Time Data Mining-based Intrusion Detection. In DARPA Information Survivability Conference and Exposition II. June 2001.

[9] Marco de Vivo, Eddy Carrasco, Germinal Isern, and Gabriela O. de Vivo, “A review of port scanning techniques”, Computer Communication Review, 29(2):41--48, April 1999.

[10] C. Leckie and R. Kotagiri, “A probabilistic approach to detecting network scans”, “In Proceedings of Eighth IEEE Network Operations and Management Symposium (NOMS 2002)”, Florence, Italy, 15-19 April 2002.

[11] Lih-Chyau Wuu Sout-Fong Chen , “Building intrusion pattern miner for snort network intrusion detection system”, Security Technology, 2003.

[12] Marc Norton, Daniel Roelker, “Snort 2:Protocol Flow Analyzer”, 6/2002

[13]SourceFire Inc., “Snort 2:Detection Revisited”, 2/2002

[14] Marc Norton, Daniel Roelker, “Snort 2:Rule Optimizer”, 6/2002

Appendix

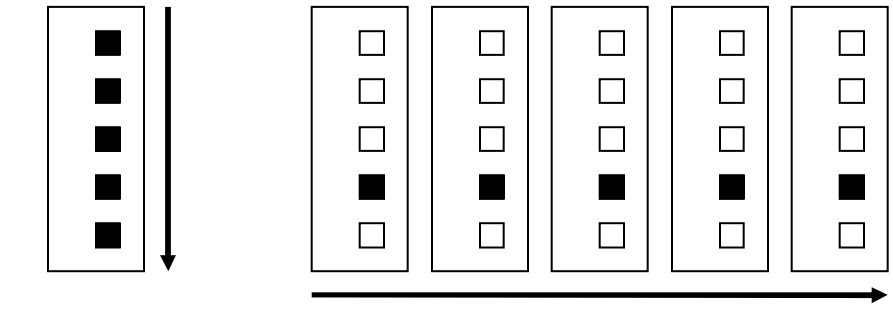
We use this section to discuss some topics which we studied as a part of our project works.

A.1. Port Scanning

Port scanning involves scanning the ports of a target system in order to find open doors to the target. Port scanning is used to get vital information of the host computer which can be used to launch and attack on the host. Thus, port scanning is a prelude to an attack and hence was an important part of our study of IDSs.

There are two main strategies for port scanning:

1. Scanning several ports of a single host
2. Scanning a specific port on a number of hosts (port sweep)



(a) Scanning a Single host

(b) Scanning several hosts for a specific

A.2. Port Scanning Techniques

A.2.1. TCP Scanning

TCP scanning uses behavior of SYN and FIN flag to identify open and closed ports. If an open port receives SYN flag, it will reply with SYN | ACK while the closed port will reply with RST. When FIN is received by an open port, there will be no reply but if the port is closed, the reply will be RST. Different type of TCP scanning techniques are:-

1. Full connection

2. SYN scanning
3. FIN scanning
4. XMAS Scanning
5. NULL Scanning
6. Indirect Scanning
7. Coordinated Scanning

A.2.2. UDP Scanning

When a UDP port is closed, it replies with an ICMP “host not reachable” message. UDP scanning uses this feature to scan hosts.

A.2.3. Ident Scanning

It exploits the identity protocol. In ident scan, a full connection is established with the host and a packet with ident request is sent to ident on TCP port 113. This will allow scanner to know which user is running the daemon on connected port.

A.2.3. Ping Scanning

Ping scanning is done either with normal ICMP echo packet or with TCP ping. In both case, the reply will give information about the port for which it was sent. If port is closed, ICMP host unreachable message is received.

A.3. Probabilistic Technique to prevent Network Scan

The technique needs three parameters to detect scanning: A) Source IP B) Destination IP C) Destination Port. There will be two lists: 1) Source IP <> Destination IP 2) Source IP <> Destination Port. This technique forms a square with 4 probability values. Any attempt to scan a particular host or a particular port will cause the probability values to go outside the square. Figure shows how the boundaries are formed from the values obtained from two lists.

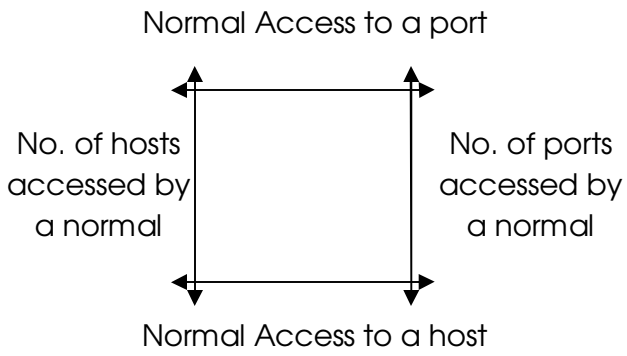


Figure 5.2

Normal access pattern of a destination IP can be found by

$$P_{norm}(d_j) = \frac{n_s(d_j)}{\sum_{all\ d_{j'} \in \mathcal{D}} n_s(d_{j'})}$$

Here, d_j is destination port and D is set of all destination IPs.

Same is the case for destination port. For a source *i*, which accessed 3 destinations $\{d_{j1}, d_{j2}, d_{j3}\}$ we will count

$$P(\mathcal{D}_i) = P(\mathcal{D}_i = \{d_{j1}, d_{j2}, d_{j3}\} \mid |\mathcal{D}_i| = 3) P(|\mathcal{D}_i| = 3)$$

i.e. probability of the selection of $\{d_{j1}, d_{j2}, d_{j3}\}$ when 3 destinations are accessed with the probability of accessing 3 host by a single source.

Same is the case for the destination port. If these probability values cross normal limits, it may be a potential scanning attempt.

B.1. Snort

Snort is a Network Intrusion Detection system, which is used to detect malicious activity in the network traffic. It is a widely used NIDS and this motivated us to study its architecture and analyze the different components of an IDS. Snort can be configured to run in four different modes i.e. as packet sniffer, packet logger, Network-based Intrusion Detection System and Inline Mode (IPS).

Sniffer mode: - In this mode Snort uses a packet capturing tool to sniff packets from the network traffic and display it on console. No logging is done in Sniffer mode.

Logger mode: - In this mode packets are logged on to secondary storage for future reference.

NIDS mode: - In this mode Snort will analyze the contents of a packet, compare them with set of pre-defined rules and generate alerts if a match is found (i.e. if a packet is found to be malicious).

Inline mode: - This mode is known as Intrusion Prevention mode. In this mode Snort will take raw data from IPTABLES and check it against its rule set. If any alerts are generated then IPTABLE rules are updated accordingly to prevent that malicious activity from occurring.

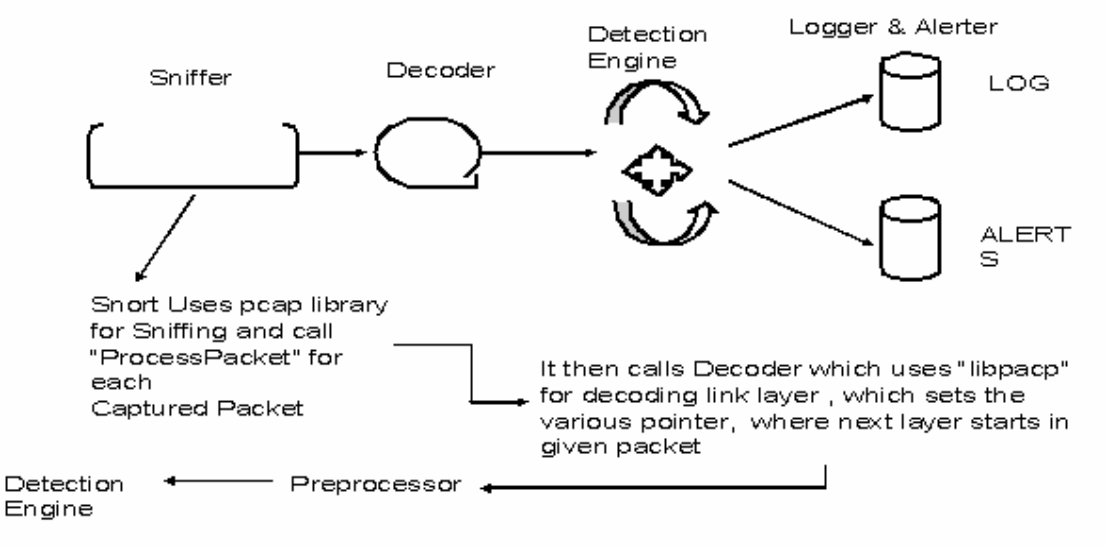


Fig: Snort Architecture

B.1.1. Sniffer

Sniffer module is used to sniff the packets from network. It uses pcap library (Packet Capture) for doing so. Once the packet is captured it is passed to the decoder for placing various pointers for future processing.

B.1.2. Decoder

Decoder will place various pointers in the packet structure to make it easy for other modules to get data of specific layer. Once the pointers are set packet is passed to the preprocessor.

B.1.3. Preprocessor

As snort is rule based intrusion detection system, it can't detect anomaly spread over multiple packets. Preprocessor is the remedy for this. Preprocessor is a separate plug-in module which can be loaded by making entry in Snort configuration file. Once snort is started with preprocessor support it will detect anomalies based on the type of preprocessor.

There are few preprocessors which we would like to discuss here.

Fraq2

This is used to detect packets having overlapping IP fragments. This vulnerability causes some machines to hang or even reset depending on the underlying operating system.

Port-scan

Port scanning can't be detected by looking at a single packet at a time. We need some past information about the network traffic to efficiently detect port scanning. This preprocessor keeps information of source IP, source port, destination IP and destination port in order to do so. It also looks at different flags in the packet to detect specially crafted packets.

Stream4

Stream4 preprocessor will combine different packets into a large single packet to analyze the whole session and find anomalies. For example, Telnet session sends each key stroke as a packet. Stream4 preprocessor combines all these packets into one single packet and passes this new packet to other modules packet for further processing and detection.

Back Orifice Preprocessor

Back Orifice is used by an attacker to remotely control a compromised system. Every message sent by attacker in this method starts with "*!*QWTY?" before encryption. Attacker chooses a password, hashes it to 16 bit number and computes XOR of every message with this hash value.

This operation can't be detected by Single rule based matching. Back Orifice Preprocessor can handle it very efficiently. It will compute every hash value (i.e. 65535 values) and calculate XOR of first 8 bytes of every UDP packet with this hashed value. If this hash matches to "*!*QWTY?" then alert is generated.

B.1.4. Flow Analyzer [11]

HTTP flow analyzer is used to distinguish HTTP client and server traffic. When we see the behavior of the HTTP Protocol we find that on an average server response accounts for 75% of the traffic and client request accounts for only 25% of the network traffic. It will be a waste of processing time to apply all rules to both HTTP server response and HTTP client request. Flow analyzer is used to divide packets into two sets namely server response packets and client request packets. The rules are also divided into these two sets. Each incoming packet is checked against rules from a specific set only. This reduces great amount of processing time.

B.1.5. Set Based Rule Detection [12]

Currently Snort has 3000 rules in its database. If we compare each packet with each and every rule then the detection engine will get overloaded and there is a possibility of missing out some anomalies (False Negative). In order to keep pace with the high performance gigabit network, Snort uses Set based rule detection methodology. When snort is started Rule Analyzer will make different rule sets (HTTP, TCP, UDP, IP) based on different protocols. Every rule belonging to particular protocol is placed in the corresponding rule set. In each rule set there are different sub groups based on there unique attributes.

Unique Parameters

TCP and UDP → source and destination port are unique in sense that they uniquely identify a packet.

ICMP → ICMP type field is unique attribute.

In a given rule set all rule having same unique attribute are placed in same subgroup.

The core idea is when a single packet is checked for anomaly it should not belong to more then one rule set. i.e. every packet must trigger one rule set to check. Once rule set is chosen (based on protocol) subgroup is selected based on unique attribute. This set based methodology provides efficient rule matching then sequential rule matching.

B.1.6. Snort Inline Mode

Till now what we saw was Intrusion prevention mode. It only alerts the administrator when anomaly was detected. It dose not take action against it. Snort inline mode can be used to as Intrusion prevention system. In inline mode snort take raw packet form IPTABLE (not from pcap library) and match it against it's rule set , if anomaly is detected then snort will tell IPTABLE to take action against it. Snort can update IPTABLE rule in order to DROP the packet.

B.1.7. Snort Rule [13]

Snort rule has two parts: Rule Header and Rule Body. Rule Header has six attribute, Source and destination address and ports, Protocol and action to be taken if rule is matched. Rule body has various other options to check such as TCP flags, search for content in payload, message to display when alerts are generated.