CHAPTER 8

INTRUSION DETECTION SYSTEM USING ECCENTRIC CLASSIFIERS

8.1 INTRODUCTION

Intrusion attacks aim at disrupting services ranging from simple bandwidth exhaustion attacks and those targeted at flaws in commercial software to complex distributed attacks exploiting specific COTS (Commercial Off-The-Shelf) software flaws. These types of attacks are not new and have a devastating effect by preventing normal operation of the victim sites. Attackers have not yet exploited the full range of vulnerabilities present in many online services especially attacks aimed at the application and data processing layer. With the rise of increasingly targeted and motivated attacks and attackers, these application level attacks will inevitably be exploited for nefarious gains.

8.2 APPLICATION-LEVEL ATTACKS

In today's world of increasingly sophisticated cyber attacks, application-level security threats are top priority with many network administrators, security consultants and Chief Information Officers (CIO). The loss of application access can cause enterprises dearly in loss revenue and employee productivity. Today's security infrastructure addresses the new way of application-layer security attacks and application abuse. The solution is layered security strategy that protects the network at all points of attack. Tools
should be uniquely positioned to provide application layer security by leveraging highly specialized deep-packet inspection capabilities.

An application-level attack targets application servers by deliberately causing a fault in the server’s operating system or applications. This results in the attacker gaining the ability in order to bypass normal access controls. The attacker takes advantage of this situation, gaining control of application, system, or network, and can do any of the following:

- Read, add, delete, or modify data or operating system.
- Introduce a virus program that uses the computers and software applications to copy viruses throughout the network.
- Introduce a sniffer program to analyze the network and gain information that can eventually be used to crash or to corrupt the systems and network.
- Abnormally terminate data applications or operating systems.
- Disable other security controls to enable future attacks.
- Bring forth compromises of data that needs to be kept private.
- Secure the resources and applications that work in a shared environment.

Every layer of communication has its own unique security challenges. The application-level communication is a very weak link in terms of security because that the application-level supports many protocols which provide many vulnerabilities and access points for attackers. All this variability makes application-level attacks very hard to defend against. In addition, application-level attacks are very attractive to a potential attacker because the information they seek ultimately resides within the application itself.
Table 8.1 Application-level Attacks and Services Affected

<table>
<thead>
<tr>
<th>Rank</th>
<th>Attack</th>
<th>Protocol</th>
<th>Affected Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LSASS Buffer Overrun</td>
<td>SMB</td>
<td>File and Printer Sharing</td>
</tr>
<tr>
<td>2</td>
<td>Generic DNS Poisoned Spoofing</td>
<td>DNS</td>
<td>Generic DNS</td>
</tr>
<tr>
<td>3</td>
<td>Nimda Incoming Worm</td>
<td>HTTP</td>
<td>Web or Email</td>
</tr>
<tr>
<td>4</td>
<td>Generic HTTP directory traversal</td>
<td>HTTP</td>
<td>Generic Web Service</td>
</tr>
<tr>
<td>5</td>
<td>Generic DNS Malformed Packet</td>
<td>DNS</td>
<td>Generic DNS</td>
</tr>
<tr>
<td>6</td>
<td>Generic Invalid Web Request Event</td>
<td>HTTP</td>
<td>Generic Web Service</td>
</tr>
<tr>
<td>7</td>
<td>Generic SMTP Malformed</td>
<td>SMTP</td>
<td>Generic SMTP Service</td>
</tr>
</tbody>
</table>

Table 8.1 enlists some of the major types of attacks, protocol medium and their affected service. The protocols are application-level protocols and these attacks target these protocols to inflict harm or gain unauthorized entry into the system and disrupt the services.

Like most of the network security problems, there is no silver bullet solution to fix the problems, however, there are many technologies and solutions available to mitigate the application-level security problems and to monitor the network to reduce its damage if an attack happens. Such many technologies have been developed at various levels of communications are as follows:

- Secure / Multipurpose Internet Mail Extensions (S/MIME)
- Pretty Good Privacy (PGP)
- Secure Hyper Text Transfer Protocol (S-HTTP)
- Public Key Infrastructure (PKI)
- Anti-Virus Systems
Secure Sockets Layer (SSL) and Transport Layer Security (TLS)

Internet Protocol Security (IPSec)

Firewall

8.3 MOTIVATION

Application-level attacks have got a reasonable amount of attention in the recent past. This is due to the following reasons:

**Non-Detectable or Preventable:** The attacks will typically not be detectable or preventable by existing security monitoring solutions. Since the attacks do not consume an unreasonable amount of bandwidth and could, in many cases, be indistinguishable from normal traffic.

**Efficient:** The attacker may not need as much resource at their disposal to successfully complete the attack. Application-level attacks target bottlenecks and resource limitations within the application and do not require many compromised “zombie” systems or a large amount of bandwidth. Furthermore, they can be targeted at the weakest link in an environment – for example if a web-farm of a hundred servers relies on a single back-office host to authenticate users, an application-level attack may be able to directly target it.

**Harder to trace:** Application-level attacks normally use HTTP or HTTPS as their transport. Proxy servers can therefore be used to obfuscate the true origin of the attacker; and many are available for an attacker to redirect his malicious traffic. Many of these proxy servers do not keep logs of connection attempts and could therefore successfully hide the true origin of the attacking host.

In order to achieve best IDS, the work with network-level, host-level has been extended to application-level. This research work aims at proposing the architecture to detect application-level attacks using a novel
approach namely Eccentric Classifier which is used to identify malicious clients and hacktivists along with pattern mining and extraction that decreases the false positive rate.

8.4 ECCENTRIC CLASSIFIERS FOR APPLICATION-LEVEL INTRUSION DETECTION SYSTEM

The goal of this work is to analyze the performance and detection rates of a novel Eccentric Classifier (EC) algorithm that utilizes fuzzy class association-rule mining method for detection of intrusions with more consistency in the application-level. This system combines the positive features of the fuzzy class algorithm and associative class mining that employs directed graph system for enhancing the detection ability and for working with mixed dataset attributes, and modified KDD CUP 1999 Dataset. The attacks usually do not cause congestion at the network level; thus, bypass the network-based monitoring system detection, and mitigation at the end system of the victim servers have been proposed.

Among them, the DDoS shield and CAPTCHA-based defenses are the representatives of the two major techniques of system-based approaches: session validation based on legitimate behavior profile and authentication using human-solvable puzzles. By enhancing the accuracy of the suspicion assignment for each client session, DDoS shield can provide efficient session schedulers for defending possible DDoS attacks. Captcha-based defenses introduce additional service delays for legitimate clients and are also restricted to human interaction services.

A kernel observation is that if attacks have been found by testing the clients in a group instead of one by one after categorizing them as groups. Thus, the key problem is how to group clients and assign them to different server machines in a sophisticated way, so that if any server is found under attack, then it can be immediately identified and filter the attackers out of its
client set. Therefore, EC theory has been applied to this security issue and specific algorithms and protocols have been proposed to achieve high detection performance in terms of short detection latency and low false positive rate. Since the detections are merely based on the status of service resources usage of the victim servers, individually signature-based authentications or data classifications are required; thus, it may overcome the limitations of the current solutions.

8.5 SYSTEM ARCHITECTURE

The various components of the proposed system, its interactions and constraints are discussed in detail in this section. The system architecture of this proposed work is given in Figure 8.1 that contains the components namely, Blacklisted IP Blocker, Testing Regimen Generator, Legitimate Reference Builder, Associative Rule Producer and Token Allocator and De-Allocator.

![Figure 8.1 System Architecture](image)
The proposed system consists of the following stages to identify the attacks at the application-level.

1. The first stage is blocking IP address that has been identified as potential attacks from entering the site. The requests are tokenized and allowed to enter into the system. There are certain constraints at this step listed below:
   
   - Due to some error legitimate clients can also generate an attack instance that should not permanently prevent the clients from using the service.
   - Ensuring client security cannot be managed from the server side. Hence the clients should ensure that the system is not used in their absence.

2. The next stage is generating the testing matrix and the analyzing algorithm in order to use the testing regimen, which gives the allocation of requests to the used servers and standby servers.

3. After, the legitimate traffic has been compared with the standby servers and the previous history, with the current traffic and then the system switches between standard mode and danger mode.

4. Then the system enters the danger mode after analyzing the THRESHOLD values. The associative rule mining block identifies the Id of the attack with the support and confidence.

5. The system then extracts the IP addresses from the allocated token and the 'Malicious Nodes' are blacklisted.
8.5.1 Session State Transfer

By deploying the detection service on the back-end server tier, the proposed scheme is orthogonal with the session state transfer problem caused by the load-balancing at the reverse proxies (front-end tier). To simplify the discussion of implementation details, the assumption is made that the front-end proxies distribute client requests strictly even to the back-end servers, i.e., without considering session stacked. The way of distributing token queues to be mentioned later is tightly related to this assumption. However, even if the proxies conduct more sophisticated forwarding, the token queue distribution has been readily adapted by manipulating the token piggybacking mechanism at the client side accordingly.

Since the testing procedure requires distributing intra-session requests to different virtual servers, the overhead for maintaining consistent session state is incurred. The motivation of utilizing virtual servers is to decrease such overhead to the minimum, since multiple virtual servers can retrieve the latest client state though the shared memory, which resembles the principle of Network File System (NFS). An alternative way out is to forward intra-session requests to the same virtual server, which calls for a longer testing period for each round, but to achieve the faster detection of attacks, the former method is adopted in this research work.

8.5.2 Matrix Generation

The testing matrix $M$, which regulates distributing which client request to which server, poses as the kernel part of this work. All the three algorithms proposed in the next section are concerned with the design of $M$ for the purpose of shortening the testing period and decreasing the false positive rate. Since the detection phase usually undergoes multiple testing rounds, $M$ is required to be regenerated at the beginning of each round.
8.5.3 Distributing Tokens

The two main purposes of utilizing tokens are associating each client with a unique, non-spoofed ID and assigning them to a set of virtual servers based on the testing matrix. On receiving the connection request from a client, each back-end server responses with a token queue where each token is of 4-tuple: (client ID, virtual server ID, matrix version, and encrypted key). Here “client ID” refers to the unique non-spoofed ID for each client, which is unchanged during the testing period (DANGER mode). The “virtual server ID” is the index of each virtual server within the back-end server. This can be implemented as a simple index value, or through a mapping from the IP addresses of all virtual servers.

The back-end server blocks out-of-date tokens by checking their “matrix version” value, to avoid messing up the request distribution with non-uniform matrices. With regard to the “encrypted key,” it is an encrypted value generated by hashing the former three values and a secured service key. A replacement token consists of the percent sign and a number.

8.6 IMPLEMENTAION DETAILS

The ultimate aim of the system is to mitigate Application level attacks and improving the false-positive rate. But these additional conditions also come into play as exciting or non-functional requirements namely service resource usage ratio and average response time.

Each legitimate client joins in and leaves the system at random times which are uniformly distributed, while the attacker threads arrive at time $t = 4 \times 30$ s and keep live until being filtered out. Both legitimate and malicious clients send requests which are with a random inter-arrival rate and
CPU processing time (workload) to the virtual servers; however, legitimate ones have a much smaller random range than that of the attackers.

Each virtual server is equipped with an infinite request buffer and all the client requests arrive at the buffers with 0 transmission and distribution delays, as well as 1 ms access time for retrieving states from the shared memory; each server handles the incoming requests in its own buffer in FCFS manner and responds to the client on completing the corresponding request. The average response time and incoming request aggregate are recorded periodically to generate the test outcomes by comparing them to the dynamic thresholds fetched from established legitimate profiles.

8.6.1 Eccentric Classifier Algorithm

The Peculiar Group Mining (PGM) algorithm uses the self-join and prune step to generate candidate $k+1$-object-groups from peculiarity $k$-object groups shown in the candidate PG function. The candidate PG function takes as argument $PG_{k-1}$, the set of all peculiarity $k$-object groups. It returns a superset of the set of all peculiarity $k$-object-groups. The functionality of the algorithm is listed as follows:

**Algorithm: PeculiarGroupMining($D$, $\zeta$)**

Input : $D$: the day-by-day behavioural dataset; $\zeta$: the minimum degree of peculiarity threshold.
Output : return the peculiarity groups in $D$: $PG$.
Algorithm:
for $i = 1$ to $m - 1$ do
  for $j = 1$ to $m$ do
    $OP_{ij} = POP(O_i, O_j , \zeta);$  
    if ($OP_{ij} = \text{Not Null}$) then
      $PG_2 += OP_{ij}$
for k = 3; PGk−1 = ψ; k ++ do
    CPGk = candidatePG(PGk−1, ξ);
for all candidate group c belongs to CPGk do
    Compute the common behavioral patterns of the candidate group c;
    PGk = {c ∈ CPGk | the length of the common behavioral pattern of c is
          no less than ξ}
end for
end for
return PG = Uk PGk

Function: Computing the peculiarity object pair
POP(Oi,Oj, ξ)
Require: Oi, Oj : the object pair in D; ξ: the minimum degree of peculiarity
         threshold.
Ensure: return the peculiarity object pair OP.
P = null;
l = 0; // the number of matched days in Si and Sj
for r = 1 to n do
    len = n − r + l;// (n − r): the length of unchecked days S in
    // Si (or Sj )
    if (len < ξ) then
        OP = null; // (Oi,Oj ) is not a peculiarity object pair
        return;
    else
        if (Si[r] ∩ Sj [r] = S ) then
            Si[r+1] = [Si[r] ∩ Oj];
            l = l + 1;
        end if
    end if
end for
\[ P[l] = \text{Si}[r] \cap \text{Sj} [r]; \]

// Fuzzy Cognitive Mappings State Comparison

end if

dend for

\[ \text{OP} = \{\text{Oi}, \text{Oj}\}; \]

return OP;

The PGM algorithm uses the object-extension strategy considering that the length of day-by-day behavioral data \( n \) is always relatively long and the degree of peculiarity threshold is high. What is more, the occurrence of the peculiar behavioral pattern in the objects of a peculiar group is relatively few (i.e., the support is low). However, the pattern-extension method generates a great amount of useless short patterns when the support is low.

The rule-based Fuzzy Cognitive Map (FCM) is more effective than simple FCM. Supporting fuzzy rules make FCM as fuzzy compatible and allow qualitative modelling. For misuse inference, multiple FCMs capture different types of intrusive behavior as suspicious events. All suspicious events generated by the FCMs impact the machine and/ or user alert levels. However, the degrees of impact are different depending on the nature of the suspicious event. For example, a suspicious event due to \text{Login\_Failure\_Same\_Machine\_Same\_User} should not impact the user alert level as much as a suspicious event due to \text{Login\_Failure\_Diff\_Machine\_Same\_User}.

**Algorithm: Eccentric Groups**

Eccentric\_Groups(\text{Records rec[100000]}, k)

create m1,m2,m3 initial mean for all the groups.
\begin{verbatim}
m1 := 0
m2 := 0
m3 := 0

select m1 random value from rec for group1.
select m2 random value from rec for group2.
select m3 random value from rec for group3.
m1 := Random(records)
m2 := Random(records)
m3 := Random(records)

create a, b,c for group1, group2, group3. // size of each group
    d1 := abs( m - rec[i] )
    d2 := abs( m - rec[i] )
    d3 := abs( m - rec[i] )

preprocess(Records rec)
{
// Remove the outliers and noise perform the transformation.
}
Al_Attack_Detector(Records records)
{
    preprocess(records)
    for each(rec in records)
        if(rec in blacklist)
            identify that as an "Attack"
        end if
    end for
    peculiar_group G = Eccentric_classify (records)
    for each(g in Group  G)
        //Add the peculiar group records into the blacklisted profile.
    Animategraph (records);
}
\end{verbatim}
8.6.2 BNF Grammar for HTTP IDS’s Rule Language

The rule generator module generates the rule by using the following Backus Normal Form (BNF) grammar.

**Rules** : 1*(variable)

**Variable** : message-line section feature operator value

**Message-line** : start-line | header | body

**Start-line** : Request-line | Status-line

**Header** : Generic-hdr | Request-hdr | Response-hdr | Entity-hdr

**Body** : HTML | XML | …

**Section** : Method | Uri | Version | Status-code | <all the generic header fields> | <all the request header fields> | <all the response header fields> | <all the entity header fields> | <all the tags in the HTML doc or other HTTP payload>

**Feature** : parameter | size | regex | occurrence

**Operator** : = | > | <

**Value** : 1*(Alpha | Digit)

**Alpha** : [a-zA-Z]

**Digit** : [1-9]

With the help of the above BNF grammar, the HTTP IDS rule is generated as below:

**#Part1: Variable Definition**

```
'VAR' body_1
body_1 := var_name var_value;
var_value := list_of(value) | value
```
# Part2: Detection Rules

'REGEX' Id body_2;

/* Id is the identifier of the rule */

body_2 := list_of(condition) | condition

condition := feature operator term /* feature refers to one field of the protocol */

operator := contain | = | in | > | < | term := value | list_of(value) | var_name

# Part3: Action Rules

'BHV' body_3

body_3 := condition ~> action argument;

condition : boolean expression action := update | log | exit | continue

8.7 RESULT ANALYSIS

The implementation of the algorithms for this system is in JAVA language as it provides numerous flexible network operations and good user interface designs. Initially a Dataset is asked to load into the system. It is real time and it includes both HTTP payloads and Dataset values for both network and application level detection.

The dataset is provided as an input to the system and the client quantum times are all processed and the threshold time limit is being verified as when the requests are being serviced. The monitor system then responds with the intermediate ALERT mode operation.

The graphical plot of the system is then generated as shown in Figure 8.2. A graph module is displayed to visually represent the internal process and the threshold limit for every request field that is being processed.
and being analysed currently by the system. The threshold value of the system for particular requests has been made dynamic so to improve false positive rates and prevent early misunderstandings. It improves efficiency, thereby.

![Figure 8.2 Initial Phase of the System](image)

The values that mark above the threshold for consecutive three cycles is definitely not allowed to compromise the system's functionality and it is put under ABNORMAL mode to test for the critical level of the threshold that has been classified for the attacks. In Figure 8.3, we simulate identifying \( d = 10 \) attackers out of \( n = 1,1000 \) clients with the number of virtual servers ranging in \( \frac{1}{2} \) to 200. It can be seen, on the one hand, that all the false negative rates are upper bounded by 5 percent and decreasing as \( K \) goes up for all the three algorithms.

This makes sense since the most possible case for an attacker to succeed hiding itself is that it is accompanied by many clients with low request rate and workloads in a testing pool. In this case, it is quite possible that the aggregate inter arrival rates and workloads in this server are still less
than that of a server with the same number of legitimate clients. Therefore, the less clients serviced by each server, the less possibly this false negative case happens.

Notice that if there is only one attacker but no legitimate client in the server whose incoming traffic is not quite dense, which changes the number of active clients that varies the current virtual server. On the other hand, the testing latencies and number of testing rounds keep declining from less than 11s and four rounds respectively, which is because the identification process becomes faster when the number of virtual servers are increased.

![Figure 8.3 Running Phase of the System](image)

A graph is plotted for the average time taken for analysing an html file, (processing time) versus the payload size as shown in Figure 8.4. As the payload size increases, the amount of the text that needs to be matched increases, and so the processing time also increases. The detection rate increases by combining HTTP header and payload (Hyper Text Mark-up Language (HTML) and Scripts).
Figure 8.4 Processing Time Vs Payload

Figure 8.5 shows a comparison of Fuzzy based Detection and Non-Fuzzy based Detection for various attacks. The graph shows the detection rate of fuzzy based detection is high when compared to the non-fuzzy based detection for attacks such as DoS, Brute Force, Directory Traversal, SQL Injection, and Cross-site Scripting.

Figure 8.5 Comparison of Fuzzy Vs Non-Fuzzy Intrusion Detection
The rule based IDS system proposed has an efficient memory usage as the size of memory needed for a working system relies on the rule table size. The IDS developed, updates the signatures and rules automatically, because of the dynamic changing nature of attacks. Fuzzy Intrusion Detection System proposed has an accurate prediction. They show better performance in terms of the detection rate and the time taken to detect an intrusion.

![Image: Figure 8.6 Variations in Response Time]

The objects in each of the protocol fields that are to be searched are plotted in Figure 8.6. It is observed that if the number of objects to be matched in each protocol field is increasing the Response time increases linearly. But the response time tends to saturate after a specific number of rules.