Université du Québec à Chicoutimi

Mémoire présenté à
L'Université du Québec à Chicoutimi
comme exigence partielle
de la maîtrise en informatique

offerte à

l'Université du Québec à Chicoutimi
en vertu d'un protocole d'entente
avec l'Université du Québec à Montréal

par

YUAN WEI

# *AN INTRUSION DETECTION SYSTEM ON NETWORK SECURITY FOR WEB APPLICATION*

août 2006

# ABSTRACT

For the last 15 years, significant amount of resources are invested to enhance the security at system and network level, such as firewalls, IDS, anti-virus, etc. IT infrastructure tends to be more and more secure than ever before. As an ever-increasing number of businesses move to take advantage of the Internet, web applications are becoming more prevalent and increasingly more sophisticated, and as such they are critical to almost all major online businesses. The very nature of web applications, their abilities to collect, process and disseminate information over the Internet, exposes them to malicious hackers. However, the traditional security solutions such as firewall, network and host IDS, do not provide comprehensive protection against the attacks common in the web applications.

The thesis concentrates on the research of an advanced intrusion detection framework. An intrusion detection framework was designed which works along with any custom web application to collect and analyze HTTP traffic with various advanced algorithms. Two intrusion detection algorithms are tested and adopted in the framework. Pattern Matching is the most popular intrusion detection technology adopted by most of the commercial intrusion detection system. Behavior Modeling is a new technology that can dynamically adapt the detection algorithms in accordance with the application behavior. The combination of the two intrusion technologies has dramatically reduced false positive and false negative alarms. Moreover, a Servlet filter-based Web Agent is used to capture HTTP request. An isolated Response Module is developed to execute pre-defined action according to the analysis result. A database is involved to provide persistence support for the framework. Also, several simulation experiments are developed for evaluating the efficiency of detecting capability.

# ACKNOWLEDGEMENT

This thesis would not be possible without the support of many people. First of all, I would like to thank my supervisor, Professor Cao Zuolang, for his great enthusiasm and indispensable guidance throughout this research. Thanks, Zhu Hui, my associated director, for sharing his knowledge and experience in web security issues. Thanks, for all your help and invaluable advice.

I would like to thank all the faculty and student members of Computer Department for their inspiring discussion and sharing their ideas. I would like to thank, in particular, Chen Tao for his help in performance testing and fine tuning the design of the framework.

This research would not have been possible without my family support. Nothing but their love, moral support, patience, and understanding give me great encouragement to go ahead.

Thanks, thanks, thanks, and more thanks to all of you!

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1

# INTRODUCTION

In this chapter, the motivation and objective are reviewed following by an overview of the skeleton of the thesis. A brief introduction of the project is included in this section.

## 1.1 Motivations

Information security is serious issue in today's extensively interconnected cyber space. Unauthorized network intrusions and computer-related fraud initiated abuses have dramatically increased due to the popularity of Internet and the implicit anonymity of network users. The commercial sectors, academic institution, government even individual desktop users are now victimized at risk from the increasing network attacks.

Since most firewalls are effective in protecting against common attacks at the network-layer, the target of attacks has changed to application-layer, where monetary return can be achieved. Meanwhile, operation system vendors have kept patching up published and unpublished vulnerabilities, so the weakness of Web application becomes the easy target of attacks.

In response to this emerging phenomenon, many solutions have been proposed to enhance application security, among which Intrusion Detection Systems (IDS) is the most effective and meaningful one.

Lacking the standardization and the supports from the operating system vendors, IDS solutions introduced their unique approaches and algorithms to detect intrusions. However, the proposal of common protocols and application programming interface are required so that the research of intrusion detection can share information and resources.

With the help of Artificial Intelligence (AI), data mining and other advanced algorithms, academic research communities attempted to develop advanced technologies to detect intrusions in large-distributed environments. However, little effort has been invested in application intrusion detection, which would be able to detect attacks targeted at business logic instead of static protocol stacks. Thus, the development of a flexible and easy-to-implement Web application-specific Intrusion Detection Framework is more desirable at present.

Additionally, with the popularity of Java technology and J2EE standard, proposal of a platform-independent Intrusion Detection Framework for Web application becomes an emerging issue. Therefore, the project is conceived and designed to provide an effective and efficient web application security framework with advanced algorithms.

## 1.2 Objective

The main objective of this thesis is to propose a common Intrusion Detection Framework for Web application, which would be able to work with any custom-built web applications. With advanced framework architecture and effective detection algorithms, this framework can process various security-related data, detect and prevent intrusion effectively with less false negative and false positive.

This J2EE-compatible framework follows the thought of modular architecture design methodology. It consists of the following four parts:

- A filter-based Web Agent is developed. The agent collects network traffic, sends the captured information to the Web Intrusion Detection System (WebIDS) for intrusion analysis, and receives instruction message from WebIDS to take appropriate actions against the network packets.
- Advanced intrusion detection algorithms, pattern matching and behavior modeling, are developed. They form the core component of WebIDS, Analysis Engine.

- Another critical component of WebIDS is Response Module. It is responsible for execution of specific actions determined by the analysis result. The module might filter out intrusion payload when an attack is detected, or send instructive messages to Web Agent.
- Database module is included in order to provide persistent storage for the WebIDS.

## 1.3 Thesis Contribution

This thesis proposes an Intrusion Detection Framework for Web application with advanced detection algorithms. The major contributions of this thesis are summarized as the followings:

- A complete Intrusion Detection Framework is proposed. With this framework, various security relevant data can be collected and analyzed by one system. And the framework is highly adaptive to allow any new intrusion detection algorithm to be conveniently deployed as plug-ins, since it adopts designing concept based on modular architecture and it is strictly compatible with J2EE development standard.
- With the introduction of Servlet Filter technology, the job of data collection becomes easier. It captures raw network traffic and takes appropriate actions against the packet based on the instructive message from Analysis Engine.
- Pattern Matching Algorithm and Behavior Modeling Algorithm are the advanced algorithms developed in this project. The combination of two algorithms decreases false negative and positive efficiently.

## 1.4 Thesis Organization

The remainder of the thesis is organized as follows.

Chapter 2 briefly describes background information. It discusses the current issues associated with the security of Web application and the reason that we have to introduce Intrusion Detection System to prevent attacks coming from outside network. Then, comparison between Intrusion Detection System and Intrusion Prevention System is reviewed. At last, relevant terminologies and information of IDS are reviewed.

Chapter 3 presents the overall architecture of IDS framework developed in this project. Relationships of the components and primary functions of each module are revealed.

From chapter 4 to chapter 7, the design details of the Intrusion Detection Framework for Web application are explained.

Chapter 4 concentrates on a design of Filter-based Web Agent. Background information on Servlet Filter technology is reviewed briefly. Then the detailed implementations are revealed.

Chapter 5 focuses on the intrusion detection algorithms and implementations. A prevalent pattern-matching algorithm is discussed; a more advanced and effective behaviour-modeling methodology follows.

In chapter 6 and chapter 7, other two modules in the framework, Response Module and Database are revealed. In chapter 6, design of Response Module is delivered. Chapter 7 presents the details about database schema and database-relevant technologies, including data access supported by Spring-framework and Hibernate.

Experiment result and evaluation of the project are the primary content of chapter 9, where several case studies are presented. Chapter 10 concludes this thesis and points out some future work.

# CHAPTER 2

# RELATED WORK

First of all, background of information security (i.e. Web application security vulnerabilities) is examined. Subsequently, the need for Intrusion Detection System can be discussed. The debate on the "IDS is dead" is discussed here too.

In the Internet era, information systems in the government and commercial sectors are distributed and highly interconnected via local area network and wide area network. These networks provide potential avenues for attacks mounted by hackers and other adversaries. Therefore, the methodology for protecting the privacy and improve security level of these interconnected computes in the Internet is a significant issue.

## 2.1 Goal of Network Security

A paper written by Donn Parker [7] outlines six elements of security that must be engraved on mind of each security administrator. We think it is worth evaluating any website by determining how it complies with these six elements.

1. *Availability:* the system must be available for use when the users need it.
2. *Utility:* the system, and data on the system, must be useful for a purpose. Similarly, each web component must have specific pre-defined function.
3. *Integrity:* the system and its data must be complete, and in an available condition.
4. *Authenticity:* the system must be able to verify the identity of users, and the users should be able to verify the identity of the system.
5. *Confidentiality:* only the owner of the data should know private data. The protected data cannot be disclosed in unauthorized fashion.
6. *Possession:* the owner of the system must be guaranteed that the system is under his control. Losing control of a system to a malicious user affects the security of the system for all other users.

## 2.2 Why We Need Intrusion Detection System

The reasons that we need IDS can be concluded into following three points:

- The inherent vulnerabilities in the traditional network security hierarchy demonstrate that it is impossible to ensure network security without any external protection.
- Firewalls cannot guarantee 100 percent security.
- The prevalent flaws in Web application also declare that the introduction of intrusion detection system is desirable.

### 2.2.1 Traditional Network Security Approaches

#### Improvement on Authentication

In aim to distinguish network machine from each other and deliver messages, a source and destination address for the network packet are required. A machine that claims to have a particular network address might not be telling the truth. An attacker can disable one of the machines and impersonate that machine using IP address impersonation [26]. Therefore, the authentication only based on address is unreliable. The open nature of the Internet also makes most of attacks possible. If packets are sent unencrypted between systems, then an adversary somewhere along the path can sniff the network and read information contained in the packets fairly easily.

To improve the security of IP packages in a network, the IPSEC (IP Security) standard was introduced. IPSEC provides two alternatives called Authentication Header (AH) and Encapsulation Security Payload (ESP). AH is sufficient to prevent impersonation of IP headers and IP address. ESP can provide privacy, integrity, or both. Naturally, IPSEC is slower than unprotected IP traffic because of the additional path lengths introduced for cryptographic computations.

### Improvement on Access Control

Network communications also require some forms of access control. It could be classified into two levels. One is application level access control, and the other is network level access control, or packet filter access control [26]. Application access control is application-oriented, and can be configured independently. Packet filter access control works at a level of the network stack and control the traffic based on the permission rules. Thus, packet filter operates at a lower network layer than the application. The biggest drawback of packet filter is that it can operate only on the fields that appear in the network packets. If access control decisions require higher-level support, these decisions are not available until they reach the receiving destination.

Firewalls are the most popular and prevalent commercial solutions protecting the network. Based on the configuration, packets meeting the criterion are forwarded. Those that fail the check are dropped. Moreover, almost every firewall today is equipped with a mechanism to provide secure IP traffic based on the IPSEC standard.

### Improvement on Encryption Techniques

Netscape® introduced SSL (Secured Sockets Layer) protocol for transmitting private documents via the Internet. SSL works by using a public key for encryption and a private key for decryption. Because SSL encryption depends heavily on keys, people normally measure the effectiveness or strength of SSL encryption in terms of key length. Now, 128-bit encryption is recognized by the most makers of Web browser.

Based on the past history of improvements in computer performance, security experts expect that the Brute-Force attack could not crack the 128-bit encryption for at least the next ten years. However, the potential threat is still there.

## 2.2.2 Firewall Is Not Enough

Now we have robust enough firewall. It provides better access control and supports more reliable IPSEC standard. However, the network is still unsecured. For example, in the year 2000 the so-called Distributed Denial of Service (DDoS) attacks blocked several major commercial sites, including Yahoo and CNN, although they were protected by firewalls. The inherent limitations that firewalls have make them insufficient for intrusion detection and prevention. These weaknesses can be summarized as follows:

- Firewalls can be compromised or bypassed, and do little to protect against attacks initiated from insider. Moreover, a hacker can easily exploit a bug that already exists in the firewall implementation.
- Many inside people use modems to connect to the outside world from the secure network, and unwanted traffic can enter through the modem connections. Thus, firewalls alone are not enough to fulfill security needs.
- Traditional firewalls are designed for improving security of network layer instead of application layer. They allow certain packets to pass through or else disable access for pre-defined data flow path. However, many of the latest infiltrations of networks occur through the firewall using the ports that the firewall allows by design or default.
- With the invention of SSL, intruders can pass right through network firewalls and go directly to the application, because they are using encrypted connections. Even deep-packet inspection is powerless to detect simple attacks delivered in this way.
- Human intervention is required to decide how to control traffic and configure the firewall to accept or deny packets. A single security policy established for the wrong reasons can lead to a system being vulnerable to outside attackers. Therefore, configuration of network firewalls is a complex and a sensitive task to Web administrator.
- Firewalls do not know what happens once the traffic gets through the firewall.

### 2.2.3 Security Vulnerabilities of Web Application

Today, there are various security vulnerabilities in Web applications, such as HTTP header, HTML, scripts, and cookies. Web-based attacks utilize web sites to send spam email that blocks inboxes, and mines confidential information. According to recent reports by Gartner [20], over 70 percent of Web attacks occur at the application-layer. Those vulnerabilities in Web applications become the primary attack targets in the network. The vulnerabilities related to the Web application can be categorized as follows:

- *IT Infrastructure Vulnerabilities:*

  Exploiting IT infrastructure vulnerabilities is probably the easiest way to attack an application. Thousands of known vulnerabilities exist in the basic components that form integrated Internet environments. Attackers, keeping themselves up to date with all published vulnerabilities, often find it is extremely easy to take advantage of them. The best well-known flaw of Web application server was in the IIS 4.0/5.0 developed by Microsoft®.

- *Software Vulnerabilities:*

  Designing and maintaining a secure web application are tedious tasks that require constant quality assurance and security analysis. Application developers often deploy third-party software and customize it to their specific needs. As a result, "holes" in the deployed software and errors created during the customization process bring serious vulnerabilities in the final application production, due to lacking of security knowledge and experience. Additionally, insecure application development patterns and practices used by developers might cause inevitable mistakes. For instance, any user can manually change hidden parameters in HTML documents and then submit the modified values to the remote server.

- *Database Vulnerabilities:*

  The database is not only the core components of most applications but also the most attractive target of attacks. Various vulnerabilities in database have been published in these years. Other than those well-known problems, SQL injection or other database-related attacks could easily delete, modify, or retrieve database records, where Web applications have access to a database directly.

## 2.3 Overview of Intrusion Detection System

From the above analysis, we have to admit that we need other instrument to guarantee the security of IT environment. So, the next question is which one we will choose to protect the Web application, Intrusion Detection System, Intrusion Prevention System, or both? Before we jump to the discussion, the terminologies related with Intrusion Detection System have to be reviewed.

### 2.3.1 Terminology

#### Intrusion

An intrusion can be defined as [Heady: 21]: *"An intrusion is any set of actions that attempt to compromise the integrity, confidentiality, or availability of a resource."*

It can also be defined as a violation of security policy [12]. In this regard, the definition of an intrusion may be different for various organizations according to their policies. For example, a login at midnight is considered as a kind of intrusion in many companies, regardless of whether the connection is from inside or outside the physical perimeters of the organization.

#### Intrusion Detection System

Intrusion detection is the process of identifying and responding to malicious activities targeted at computing and networking resources.

An Intrusion Detection System, or IDS for short, helps computer systems deal with or prepare for attacks by analyzing gathered information of security problems. An Intrusion Detection System usually includes three components: data collector, data analyzer and responder.

A data collector captures security-related data such as system logs, network packets, audit data, etc. A data analyzer examines if violation of security policy or exploitation occurs. Upon detecting such exploitation, the responder component triggers an alarm, takes appropriate actions, and saves the evidence of the intrusion for further analysis.

In short, Intrusion Detection System cannot prevent the malicious attacks, but it can help administrator detect intruders when they enter Web application. Thus, IDS can help stop hackers before they get too far.

### 2.3.2 IDS or IPS

Now let us face the debate on Intrusion Detection System sparked by the assessment of Gartner [20]. The report declared that intrusion detection systems would be dead and predicted the market for such products would be gone by 2005. Moreover, the report also announced that Intrusion Prevention System (IPS) is the answer to most security issues.

Intrusion Prevention System, or IPS for short, is an active intrusion prevention system. It can detect malicious information within normal network traffic and block the offending traffic automatically before it does any damage rather than simply raise an alert.

Thus, the biggest difference between passive IDS and IPS is that once malicious activity is detected, the IPS has the ability to take active defensive actions. However, we do not think IPS will take over passive IDS, because the following reasons:

1. It is really difficult to locate the position of passive IDS in the network. But IPS, just like firewalls, is relatively easy to find.
2. Suppose we need more detailed network activities to do correlation. With an IPS we have to consider thoroughly, because performance can be an issue if we want to get a huge amount of these information. If we choose passive IDS, we have less risk of performance impact.
3. Since an IPS combines the blocking capabilities of a firewall with the deep-packet inspection of IDS together, the configuration of an IPS is very difficult. But IDS is more flexible to make changes fast.

Moreover, IPS does not have any advantage over active IDS. Today, many of the IDS vendors are adding active response capabilities to their products. The concept behind this strategy is that the IDS can detect an attacker and then move to stop his attack. An active IDS responds to the suspicious activity by logging off an intruder or by reprogramming the firewall to block network traffic from the suspected malicious source [9]. In the other word, both active IDS and IPS have the capability to prevent the attack from being successful.

Additionally, a lot of people support the point that there is no conflict between IDS and IPS. Use an IPS to prevent what can reliably be prevented, use an IDS to detect the more difficult to prevent attacks and collect additional forensic data.

According to the comment stated above, we do not believe IPS is the way of the future. And we do not think IPS will take over IDS, as IPS does not bring more extra functionality. Actually, both IDS and IPS share common problems, i.e. signature update. However, it is important to remember that no single security device could stop all attacks all the time.

## 2.4 Classification of IDS

At present, from the techniques that a system uses, intrusion detection systems can be categorized into two broad classes: anomaly detection system and misuse detection system [10] [15]. From the domain that a system protects, intrusion detection systems can also be categorized into three classes: host-based Intrusion Detection System, network-based Intrusion Detection System and Hybrid Intrusion Detection System.

### 2.4.1 Host-based IDS, Network-based IDS and Hybrid IDS

**Host-based IDS**

Host-based IDS employs the host's audit trail (such as audit logs, application log, system information) as the main source of input for detecting intrusions. HIDS can be installed on many different types of devices, such as servers, workstations and notebook computers. Although the HIDS is limited in scope and cannot detect simultaneous attacks against multiple hosts, it can be powerful tool for analyzing a possible attack by recording what the attacker did. A HIDS usually provides much more detailed and relevant information than a NIDS.

### Network-based IDS

The traditional HIDS was designed to detect intrusions in a single host. As the focus of computing shifted from mainframe environments to distributed networks of servers, it has considered intrusion not only of single hosts but of networks as well. NIDS builds its detection mechanisms to monitor network traffic. It can be installed on active network elements, for example on routers. NIDS utilizes the source and destination IP address to deduce security-related parameters, like the number of total connection arrivals in a certain period of time, the number of packets to/from a certain machine, or the arrival time between packets. These parameters can be used to detect port scans or DoS (Denial of Service) attempts.

### Hybrid IDS

Hybrid IDS is a combination of Network-based IDS and Host-based IDS. Thus, Hybrid IDS would monitor network traffic, and monitor the host sources that a HIDS would. For example, Prelude [18] is a typical open source hybrid IDS framework.

### 2.4.2 Misuse Intrusion

Misuse Detection [22] (or Signature Detection) attempts to encode knowledge about attacks as well-defined patterns and monitors for the occurrence of these patterns. Signatures are patterns for detecting known attacks or misuse symptoms. "*Individual patterns can be composed of single events, sequences of events, thresholds of events, or general regular expressions in which AND and OR operators are allowed [26].*" They may be simple as in the case of character string matching a single term or command, or complex as in the case of state transition written as a formal mathematical expression. One technique used to satisfy this may be having rules that describe the system state changes, i.e. STAT [13].

The rule-based system or signature-based system monitors the system resources and logs to match attack signature. When an attack is detected, an alarm is triggered. For example, an attempt to exploit a XSS (Cross-site Scripting) intrusion can be caught by examining if there is JavaScript or HTML in the parameter field. This can be accomplished using a pattern matching approach. Therefore, the accuracy of the misuse intrusion detection is considered good, but its completeness requires that the attack knowledge base should be updated regularly. In the other word, misuse intrusion detection usually has low false positive, but high false negative. At recent, SNORT [19] becomes the most popular open-source, signature-based network intrusion detection system.

In the Section 5.1, we discuss a Pattern Matching Engine. It follows the nature of misuse detection technique, and realizes a pattern-matching algorithm.

**Advantages of Misuse Detection**

- Misuse detection concerns only the system data items related to pre-defined pattern, does not need to exhaustively analyze all system events, thereby reducing system overhead. Meanwhile, administrator can choose customized patterns for different Web applications.
- Misuse detection matches system events with clearly defined patterns of vulnerabilities and exploitations. This technique is very efficient and effective to detect well-known attacks.

**Disadvantages of Misuse Detection**

- It is hard to collect all the required information for detecting all known attack, and keep it abreast with new vulnerabilities. The construction of signature database is a time-consuming process and prone to mistake. And it is the most critical drawback of misuse intrusion detection.
- The misuse detection approach can be highly accurate, but it cannot detect intrusions that fall outside its predefined list of rules describing known vulnerabilities and exploitations.
- A complicated intrusion scenario is very difficult to abstract for generating accurate intrusion signatures. In addition, it is extremely difficult, even impossible to construct intrusion signatures to accommodate all variants of intrusion scenarios. This is the reason that causes the misuse detection generates low false positive, but high false negative.

### 2.4.3 Anomaly Detection

Anomaly Intrusion Detection is a prevalent approach, which is based on the detection of the anomalous behaviour or the abnormal use of computer resource [2].

A profile that describes the normality of the monitored system and/or users is always required for anomaly detection. Anomaly detection systems, for example, IDES [30], flag observed activities that deviate significantly from the established normal profiles as anomalies or possible intrusions.

For various kinds of subjects in an anomaly detection system, such as sessions, users, groups, programs and network traffics, a number of measures and attributes are used to describe the normality. Depending on the source of these input data, anomaly detection is divided into host-based anomaly detection and network-based anomaly detection [4].

Network-based anomaly detection focuses on the packets that are sent over the network and monitors the flow of packets. Source and destination IP addresses, connection start and end time are parameters used to summary network traffics.

Like host-based IDS, host-based anomaly detection concentrates on activities at hosts. Host-based anomaly detection works by establishing "profiles" of typical network activities, such as login time, number of failure logins, CPU usage, etc. An anomaly detection system uses these profiles to monitor current user's activity and to compare them to detect anomalous behaviour. Whenever a user's current activity deviates from profile significantly, the activity is considered as a possible or potential attack.

In the Section 5.2, we propose a Behaviour Modeling Engine. It is an implementation of a host-based anomaly detection system. A behaviour-modeling algorithm, which extends anomaly detection technique, is developed in the engine.

**Advantages of Anomaly Detection**

- The most significant advantage of anomaly intrusion approach is the ability to detect novel attacks against variants of known attacks, and deviations from normal usage of programs, regardless of whether the source is a privileged user or an unauthorized external user.
- The anomaly detection technique has the capability to determine the legitimate profile according to user activities or program activities without any intervention of human security expert.

**Disadvantage of Anomaly Detection**

- The high false alarm rate is generally cited as the main drawback of the statistical anomaly detection. The reason for this is that the entire scope of normal behaviour of a computer system or user may not be covered during the learning period. Also the constantly behaviour changes make it difficult to accurately grasp the condition of a normal environment in real time.
- To the statistical anomaly detection, it is difficult to determine a threshold, which is a value to evaluate if the activity should be classified as an intrusion or a normal action. If the threshold is too low, it may generate many false positive alarms; on the contrary, if the threshold is too high, the number of false negative raise. It is relatively easy for an intruder to trick the statistical analysis unit into accepting malicious attack as normal activity by gradually varying his actions over time. Consequently, setting thresholds for indicating intrusive events requires experience.
- The anomaly intrusion detection method only identifies activities as anomalies or determines the current system is in anomalous status; it cannot distinctly indicate what happens to the system or what the hacker has done to the system. The shortcoming results in a high false positive rate.

Since every intrusion detection technique has advantages and disadvantages, Intrusion Detection System proposed in the thesis adopts a hybrid approach to detect attacks or exploitations. A Pattern Matching Engine can drop a request if it matches well-known malicious attack pattern, and a Behaviour Modeling Engine can drop the input if it falls outside the normal profile.

# CHAPTER 3

# INTRUSION DETECTION FRAMEWORK

In this chapter, the proposed Intrusion Detection Framework is discussed, followed by a short description of overall development approach. Finally, relationships among components in the proposed Intrusion Detection Framework are analyzed. Each component in this framework will be described with more details in the following chapters.

## 3.1 Description

The proposed Intrusion Detection Framework in this project is in compliance with the Common Intrusion Detection Framework issued by CIDF [5] completely.

- Filter-based Web Agent is used to collect raw HTTP request (note that only request information is collected and analyzed).
- The intrusion detection approaches including Pattern Matching and Behavior Modeling are designed and implemented in Analysis Engine.

- An isolated Response Module is developed to carry out responding actions.
- A Database is involved to provide persistence to the entire system.

The research on the Intrusion Detection Framework is completed in J2SE 1.5.0 environment. Some modern development tools and technologies, like Tomcat, Spring-framework and Hibernate are employed to meet the requirements of design and programming.

## 3.2 Relationships of IDS Components

The framework of Intrusion Detection System can be illustrated with the following figure:

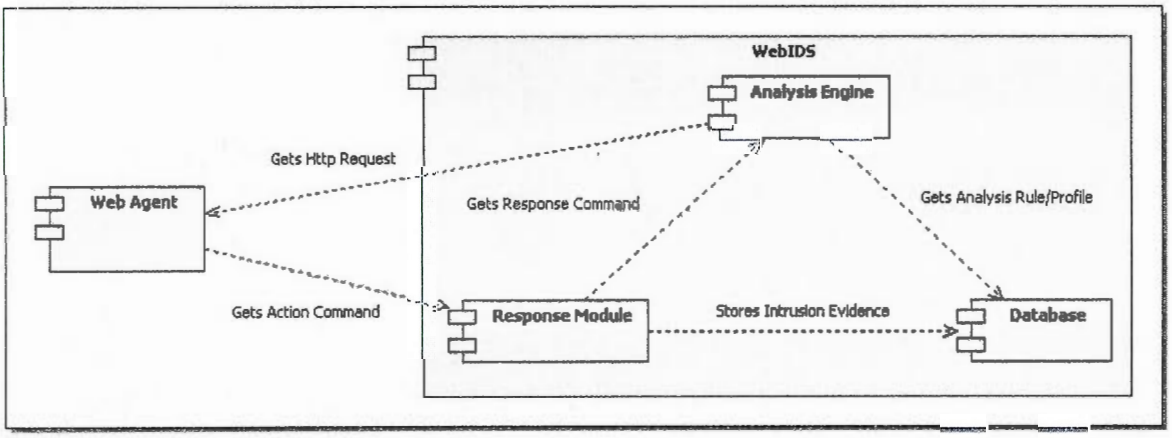


**Figure 3-1 Relationships of Intrusion Detection Framework Components**

### Web Agent

This component could be integrated with target Web application as a pluggable module. It has the following features:

- Gathers security-related raw HTTP traffic. Stores the collected data in the XML format.
- Sends the collected data in XML to Analysis Engine.
- Executes responding action (drop, pass, etc.) according to instruction received from Response Module.

### Analysis Engine

Once Analysis Engine receives the collected request from Web Agent, it analyzes if there is malicious content in the request or not. Since some analysis patterns or profiles might be stored in the Database, Analysis Engine retrieves them from the Database during the period of initialization or at runtime.

### Response Module

According to the analysis result concluded in the Analysis Engine, Response Module carries out pre-defined response actions. If an attack event is found, the collected data with malicious attacks will be inserted into the Database as intrusion evidence for further analysis and reporting. And an alarm must be raised to draw administration's attention. Moreover, instruction to drop the malicious request must be return to Web Agent.

### Database

Database component provides persistent storage for entire Web Intrusion Detection Framework. As the illustration in Figure 3-1, Analysis Engine retrieves configuration and profile information from database for intrusion analysis, and Response Module saves malicious request to database as security log.

# CHAPTER 4

# WEB AGENT

In this chapter, implementation details of Web Agent are discussed. Since Web Agent is developed with Servlet Filter technology in J2EE environment, some background related with filter technology are mentioned briefly. Second, a draft on a centralized character filtering technology is introduced, with the consideration of that attacks could bypass intrusion detection system by utilizing various character encoding. Finally, a method to normalize HTTP request data into an analyzable format is revealed.

## 4.1 Description

Web Agent is a data collector. All HTTP data to be analyzed in the Analysis Engine is captured by Web Agent. Thus, Web Agent is designed with following principles:

- Gathers security-related data.
- Provides a centralized character set filter for all input stream.

- Stores the normalized HTTP request to a file in XML format.
- Sends the XML file to Analysis Engine of WebIDS.
- Gets instruction from Response Module in WebIDS and executes correspondent action (i.e. drop the user request or deliver the request).

Web Agent gathers security-related data, including network packets data and system-related data. Network packets are the data transmitted through the network, which are then collected by the network traffic monitor or sniffer. Generally speaking, network packets data involves such information like source and destination address, source and destination application port numbers, types of packet, options of the protocols, and the content of the packets. The network packets data play an important role in Intrusion Detection Systems, since they provide detailed and valuable information of network activities.

In the Intrusion Detection Framework covered in the thesis, not only the client request is normalized, but also some system-related data (i.e. *applicationID* and *agentID*) are included in the XML files for distributed deployment scenario. Association relationship between Web application and Web Agent is very important. For example, one Web Agent could provide data collection service for multiple applications deployed in the same web server. This information should be associated accurately when administrator deploys Intrusion Detection System to specific application, so that the Analysis Engine could differentiate web applications being monitored.

## 4.2 Considerations of Web Agent

A number of design requirements for Web Agent have been taken into account in advance. They are summarized as following three aspects:

**Modularization:**

The primary task of Web Agent is to gather network data, send the collected information to WebIDS where further intrusion analysis would be carried out. And it must execute some actions (i.e. drop or pass) according to the action instruction received from WebIDS. From the view of functionality, the job of Web Agent is associated with Web application closely. From the view of modularization, Web Agent should be an isolated component, which can be embedded in the Web application easily.

**Normalization:**

The volume of network traffic is extremely huge, and it includes a lot of irrelevant information. Some measures must be taken to reduce the size of these data and eliminate non-security-related information before Web Agent sends them to Analysis Engine. With these considerations in mind, a centralized input filter routine is adopted. A suitable canonical form must be chosen and all user requests should be standardized into that form before any intrusion analysis is performed. Moreover, in order to meet the requirement of further information processing, it is necessary to normalize the information into a common format.

**Efficiency:**

In order to meet the requirements of network traffic capturing and real-time detection, Web Agent must collect network packets efficiently with the least system overhead and time consuming.

## 4.3 Filter-based Web Agent

In this project, a filter-based Web Agent is employed to capture HTTP request and receive response from WebIDS.

### 4.3.1 Feasibility Study

**Serlvet Filter**

Servlet Filter is a useful and important technique, which is introduced since Servlet 2.3, and it can be defined as follow:

*"A filter is an object that can transform the header and content (or both) of a request or response. [27]"* It is a web component that intercepts requests and responses, or manipulates the data that is being exchanged between client and server. *"As a result, the critical difference that makes filters different from other web components in that filters usually do not themselves create a response. [27]"* It provides a modular, object-orientated mechanism for encapsulating common tasks into pluggable components that are declared via a configuration file and processed dynamically [14].

### Advantages of Filter-based Web Agent

According to the features stated above, we draw the conclusion that Servlet filter is sufficient to build an efficient Web Agent. The HTTP request can be captured when it pass through the filter. After intrusion detection analysis being performed on the request data, the reconstructed response is returned to the user. The filter-based Web Agent has several important advantages:

- Filter-based Web Agent is designed to be a pluggable component within web application. It allows the seamless integration of Web Agent with the web application. By encapsulating application-processing logic into a single component, filter-based Web Agent defines a modular component that can be easily added to and removed from the target application without any code modification.
- Filter-based Web Agent is a lightweight solution. It can be invoked by the Servlet container to perform data gathering functionality without any impact to other Web components.
- By means of the modular design of a filter's implementation class and flexible filter mapping configuration, a filter can be mapped to any number of web resources in a web application. Therefore, filter-based Web Agent provides customized data gathering service for different web pages.
- Filter-based Web Agent is platform-independent as long as a Servlet container is available. This feature allows it to be easily deployed in any compliant J2EE environment.

### 4.3.2 The Implementation

The following class diagram illustrates the relationships among classes contained in the Web Agent component. Where, the job that *normalization* package defines is to normalize HTTP request. We will discuss it in the Section 4.5. Moreover, the detail about the centralized input filter, *Decoder*, is going to be reviewed in the next section.

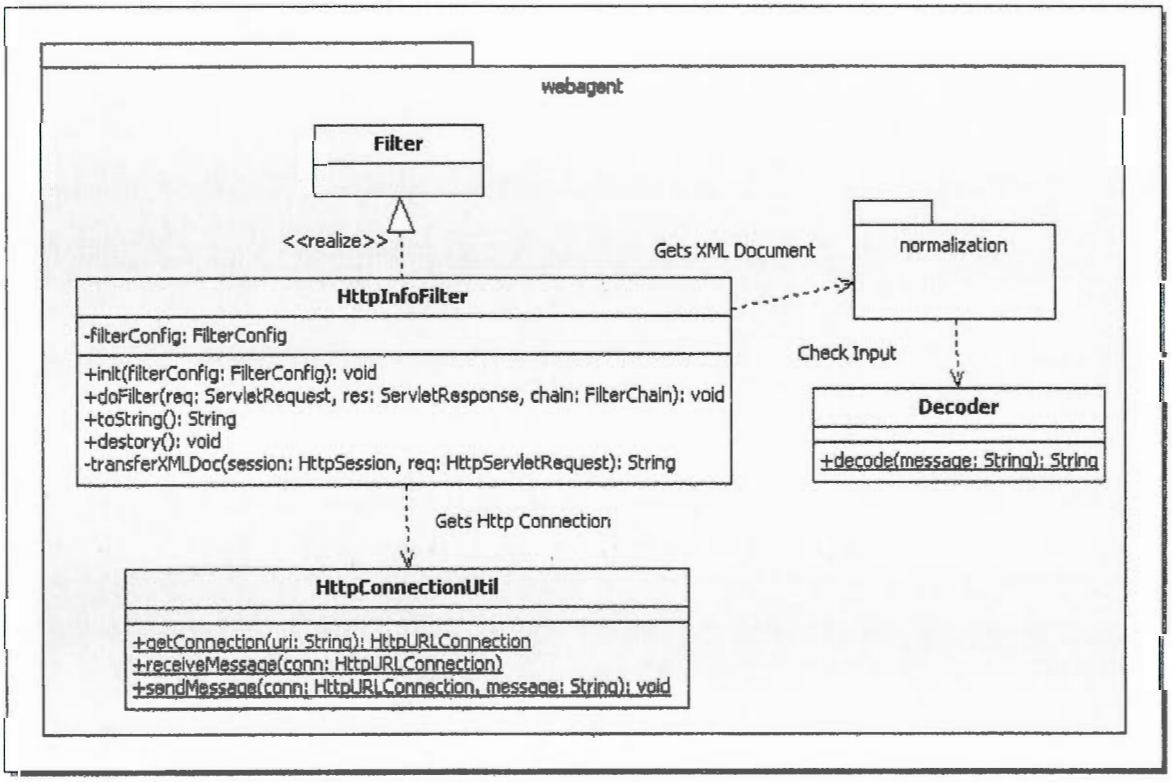


**Figure 4-1 WebAgent Class Diagram**

The implementation of filter-based Web Agent focuses on a Sevlet Filter component, *HttpInfoFilter*. It uses the functionality offered by other objects to perform data collection and normalization task. The programming details are shown in the following figure, Figure 4-2.

```
1.  // Web Agent sends XML document to and receives message from WebIDS
2.  String receivedMsg = this.transferXMLDoc(session, request);
3.
4.  // Web Agent's request does not pass analysis of WebIDS
5.  if (!receivedMsg.equalsIgnoreCase("OK"))
6.  {
7.      ...
8.      // Create a writer from an output stream that writes to this response.
9.      PrintWriter out =
10.         new PrintWriter(new BufferedOutputStream(response.getOutputStream()));
11.     // Output error comment
12.     out.println(receivedMsg);
13.     out.flush();
14.     out.close();
15. }
16. else // Otherwise
17. {
18.     ...
19.     chain.doFilter(request, response);
20. }
```

**Figure 4-2 Programming Details of doFilter()**

In line 2, a private method *transferXMLDoc* is invoked. This method requests the service provided by *normalization* package to get standardized XML document. Then it sends the data to WebIDS with the support of HTTP connection provider, *HttpConnectionUtil*. Also *transferXMLDoc* is designed to receive the action instruction from WebIDS when intrusion analysis decision has been made. The left part of *doFilter* (lines 4-20) executes the received action intrusion. If no intrusion is found or the received instruction is "OK", client will receive his/her requested page. Otherwise, client might receive a customized error page.

## 4.4 Centralized Character Filter

The responsibility of a centralized character filter is to process system-sensitive and security-sensitive characters. The activities that centralized character filter performs include:

- Canonicalize the sensitive character from one form to another.
- Filter out illegal character sequences.

Moreover, it has to be designed as a centralized component. In the other word, all forms of HTTP request must be processed by this filter. And only the HTTP request after being filtered by this character filter can be utilized as input stream for further processing. So when a HTTP request is captured, it will be canonicalized once and then utilized by other WebIDS components.

### 4.4.1 Why a Centralized Character Filter Is Needed

The reasons that we need centralized character filter include the following two points:

- A character sequence might have specific meaning at different processing points. Since we store HTTP request in the XML format, some characters (i.e. less-than sign "<") are sensitive characters we have to consider.
- Some attacks are based on the variant of character encoding. Recognition of the malformed character format is the most efficient method against these attacks. Moreover, generation of attack pattern could be much easier if the captured HTTP request could be canonicalized into a uniform form.

### Character Encoding and UTF-8

In order to use a standard and canonical character set to represent all available characters, people perceive the concept of character encoding. Now there are two popular character-encoding standards, the Unicode Standard [31] and the International standard ISO 10646 [32].

The Unicode Standard defines three encoding forms that allow the same data to be transmitted in a byte, word or double word oriented format (i.e. in 8, 16 or 32-bits per code unit). Where, UTF-8 is popular for HTML and becoming a dominant method for exchanging international text information through network.

All three encoding forms that the Unicode Standard defines can encode the same common character collection and can be transformed into one another efficiently without loss of data. However, the advantage of compatibility also makes the UTF-8 be an exploitable security flaw. The following cases demonstrate how hackers initiate attacks aiming at UTF-8 flaws.

### Case Study

Take Directory Traversal as an example, a typical input exploitation, for example. The "../" character sequence is not guaranteed to conquer, especially since most security checks get along just fine with raw character without encoding.

| UCS Code (Hex) | Binary UTF-8 Format |
|---|---|
| 00-7F | 0xxxxxxx |
| 80-7FF | 110xxxxx 10xxxxxx |
| 800-FFFF | 1110xxxx 10xxxxxx 10xxxxxx |
| 1000-1FFFFF | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx |

**Table 4-1 Legal UTF-8 Sequences**

### Overlong UTF-8

Firstly, let us consider the various representations of "."(ASCII 2E). According to the specification listed in the Table 4-1, suppose we use the second UTF-8 range (2 bytes) to represent it, we get an overlong representation, %C0%AE. Likewise, there are more overlong representations with the other UTF-8 encoding: %E0%80%AE, and %F0%80%80%AE. For the character "/" (ASCII 2F), we can also deduce its deformed representations: %C0%AF, %E0%80%AF, %F0%80%80%AF. Thus, for the character sequence "../", we might have 4*4*4 = 64 different choices to present it.

### Illegal UTF-8

We even can have more than that. Consider the representation %C0%AE of character ".". Just like UTF-8 encoding requires, the second octet has "10" as its two most significant bits. Now, it is possible to infer 3 variants for it, by enumerating the rest of the alternative 2 bit combinations ("00", "01" and "11"). Since some UTF-8 parsers simply check the least significant 6 bits (they ignore the most significant 2 bits), they would think these variants as identical to the original symbol. It should be kept in mind that these character sequences are illegal, since they do not comply with the legitimate UTF-8 formats listed in the Table 4-1.

Now, we have several diverse representations of one character sequence "../". Suppose, one of the security checks searches for "../", and it is carried out before character filter takes place, it is possible to exploit the Directory Traversal flaw with the overlong UTF-8 encoding. The reason why the attack can be successful is that it is impossible for security check to recognize all variant formats. It is the reason that drives us to introduce the centralized character filter. Thus, a suitable canonical form should be chosen and all HTTP requests must canonicalized into that form before any security checks are performed.

### 4.4.2 Countermeasure

In the implementation of centralized character filter, we use following methods to handle system-sensitive or security-sensitive characters.

In the Web Agent, the HTTP request after normalization is kept in the XML format, so we must pay attention to XML-sensitive characters. Because most character formats have an escape sequence to handle this case, we use the corresponding escape value to replace the sensitive character. As for the character encoding related to the problems, two cases have to be considered:

- If the UTF-8 encoding is illegal, filter out these character sequences.
- If the UTF-8 encoding is valid but overlong, a canonical UTF-8 form (2 bytes) is adopted as standard form. Any overlong character sequence must be converted into 2-byte-long format.

## 4.5 Normalization

After HTTP request passes through centralized character filter, the filtered and canonical data is available. Before Web Agent sends the data to WebIDS, it must convert the filtered information into a specific format for the subsequent processing.

XML (Extensible Markup Language) is considered as a desired form of the data. The reasons for adopting XML can be summarized as follows:

- Unlike HTML, XML tags identify the data itself rather than specify how to display it. It's common to use XML as a medium for data exchange on the web.
- J2EE platform provides various methods for processing XML, such as SAX and DOM, etc. Moreover, XML can be easily deployed in any J2EE-compliant environment.

Therefore, we use SAX parser and XSLT to convert HTTP request to XML format. Figure 4-3 illustrates the DTD (Document Type Definition) of XML file after normalization.

## 4.6 Conclusion

In this chapter, details about the design of the Web Agent are discussed. The Web Agent collects and normalizes security-related data. Then it sends the normalized data to WebIDS for further security analysis. It also receives instruction from WebIDS. The received instruction message determines the client request is a normal request or an abnormal intrusion and what actions (i.e. drop or pass) should be taken.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT HttpData (App, Agent, Client, Header, Session, Cookie?, Request?)>
<!ATTLIST HttpData rr CDATA #REQUIRED>

<!ELEMENT App (AppName, AppIP, AppPort, AppTimestamp)>
<!ELEMENT AppName (#PCDATA)>
<!ELEMENT AppIP (#PCDATA)>
<!ELEMENT AppPort (#PCDATA)>
<!ELEMENT AppTimestamp (#PCDATA)>

<!ELEMENT Agent (AgentName, AgentID, ApplicationID)>
<!ELEMENT AgentName (#PCDATA)>
<!ELEMENT AgentID (#PCDATA)>
<!ELEMENT ApplicationID (#PCDATA)>

<!ELEMENT Client (ClientName, ClientIP, ClientPort)>
<!ELEMENT ClientName (#PCDATA)>
<!ELEMENT ClientIP (#PCDATA)>
<!ELEMENT ClientPort (#PCDATA)>

<!ELEMENT Header (HeaderParam+)>
<!ELEMENT HeaderParam (Name, Value)>

<!ELEMENT Session (SessionID, Created, LastAccessed)>
<!ELEMENT SessionID (#PCDATA)>
<!ELEMENT Created (#PCDATA)>
<!ELEMENT LastAccessed (#PCDATA)>

<!ELEMENT Cookie (CookieParam+)>
<!ELEMENT CookieParam (Name, Value)>

<!ELEMENT Request (RequestParam+)>
<!ELEMENT RequestParam (Name, Value)>

<!ELEMENT Name (#PCDATA)>
<!ELEMENT Value (#PCDATA)>
```

**Figure 4-3 DTD for HTTP Request**

# CHAPTER 5

# ANALYSIS ENGINE

In this chapter, details about Analysis Engine are revealed. In this project, two analysis algorithms are employed: one is Pattern Matching, which is based on Misuse Detection Algorithm; the other is Behaviour Modeling, which is based on Anomaly Detection.

## 5.1 Description

Analysis Engine is the most important part of Intrusion Detection Framework for Web application, which implements of two algorithms: Pattern Matching Engine and Behavior Modeling Engine. Some considerations have to be taken into account when design Analysis Engine, and they can be summarized as follows:

- Initialization of analysis-related data, which contains configuration files.
- Pattern Matching Engine and Behaviour Modeling Engine perform intrusion analysis on the data captured by Web Agent.
- Trigger Response Module to carry out specific action once the analysis result is concluded.

## 5.2 Pattern Matching Engine

Pattern Matching Engine makes use of an extension of Misuse Detection technique [22], Pattern Matching algorithm. It attempts to encode knowledge about well-known attacks as patterns and monitors for the occurrence by matching the captured data with pre-defined patterns. If current HTTP request contains any intrusion pattern, an intrusion is identified.

**Design of Pattern Matching Engine**

- Three configuration files in the XML format, including rule-mapping file, rule definition files and attack target file must be initialized before pattern-matching analysis starts.
- According to the declarations in the target file, classify HTTP request sent from Web Agent into several parts.
- With the combination of rule-mapping file, rule definition files and target file, pattern-matching analysis can be executed in a loop.
- Whenever an analysis result is concluded, Pattern Matching Engine triggers Response Module to execute actions.
- The loop of rule matching must be terminated if an intrusion is detected or the current HTTP request completes the pattern-matching analysis.

### 5.2.1 File Description

Parameters of pattern-matching analysis have to be initialized in advance, before intrusion detection starts. Three configuration files are defined in the Pattern Matching Engine; they are rule definition files, rule-mapping file and target file, respectively. The data in these three files has to be read and mapped to corresponding Java objects, so future pattern-matching analysis can retrieve these required data easily and conveniently from system memory.

**Rule Files**

There are two types of rules defined in the Pattern Matching Engine. Where, system-rule file specifies the patterns extracted from well-known attacks; and user-rule file keeps custom rules that a Web application administrator has to declare. Table 5-1 illustrates the structure of a rule.

| Item Name | Description |
|---|---|
| ID | Rule ID |
| RuleName | Name of a rule |
| Version | Version information of a rule |
| Target | The HTTP request that a rule matches against |
| Pattern | Pre-defined pattern extracted from well-known attacks |
| CaseSensitive | A rule is case-sensitive or not |
| Length | Length of parameter defined in the Target item |
| ReferenceNo | Reference information of a rule |

**Table 5-1 Structure of a Rule**

### Rule-Mapping File

A configuration file, rule-mapping file is defined in the Pattern Matching Engine. The primary function of this file is to build mapping relationship among application, agent, rule and related action data.

Since Web Agent might associate with more than one Web applications, the mapping relationship among them must be clearly stated. For a particular combination of Web application and Web Agent, response actions to take should be determined. Therefore, rule-mapping file is a map to demonstrate relationships among rules, actions, Web applications and Web Agents. The following table explains the structure of a rule mapping.

| **Item Name** | | **Description** |
|---|---|---|
| ApplicationID | | ID of Application |
| AgentID | | ID of Web Agent |
| RuleID | | ID of Rule |
| forward | name | Two choices for this attribute of *forward* element:<br>match: HTTP request fits the rule determined by *RuleID*<br>unmatch: HTTP request does not fit the rule determined by *RuleID* |
| | actionID | ID of action to be executed |
| | actionParam | The customized action parameter |

**Table 5-2 Structure of a Rule Mapping**

### Target File

The information defined in the target file represents the part of the HTTP request data that may be used by attacker to deliver malicious content. The term "target" is used to represent various sections of HTTP protocol. Six "targets" are defined in the "target" file; they are Header, URI (Uniform Resource Identifier), Session, Cookie, Post parameters and Get parameters, respectively. The "target" defined in the rule files must be one of the six "targets". Thus, each incoming HTTP request must be broken into these six parts defined in the target file. The structure of a target and can be illustrated by the following table.

| Item Name | Description |
|---|---|
| TargetName | Name of Target |
| TargetItem | A target might contain several distinct items, i.e. HTTP Header. So this field lists possible target items that a target includes. |

**Table 5-3 Structure of a Target**

### The Relationship of Three Files

Figure 5-4 illustrates the relationship of the three files, rule definition file, target file and rule-mapping file.

- A *RuleID* is specified in the file, rule-mapping.
- According to the reference to *RuleID*, the specific rule can be retrieved from the rule definition file.

- Integrate the pattern (regular expression), case-sensitive and length arguments together, an attack pattern is generated.
- The combination of "target" reference in the rule definition file and target definition in the target file determine which section of HTTP protocol must be examined.
- The corresponding action data are retrieved from the rule-mapping file, once the analysis result is concluded.

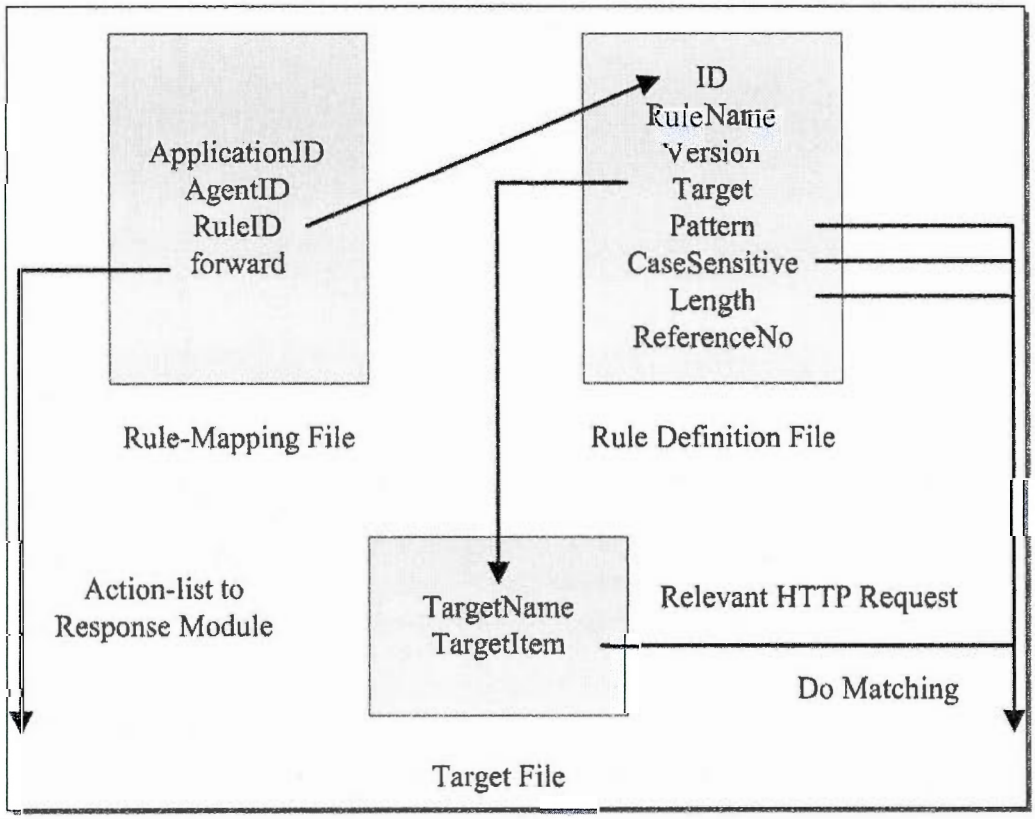


**Figure 5-1 Relationship of Tables in Pattern Matching Engine**

### 5.2.2 How to Perform Rule Matching

#### Basic Algorithm

Rule matching is the most critical part in the Pattern Matching Engine. For each specific Web application and Web Agent, several different *RuleMappings* might be defined. The *RuleMapping* traversal involves following steps:

1. If there are un-visited *RuleMappings* left or special flag (i.e. *deny-flag, pass-flag*) is not set, the loop continues; otherwise it terminates.
2. Retrieve current *RuleMapping* from rule-mapping file, and the rule to be used can be determined based on the conditions defined in the rule.
3. Once the rule is selected, "target' of the HTTP request data is determined through the *target* element declared in the rule definition file and reference to the target file.
4. According to the "target" specified in the step 3, retrieve the corresponding HTTP section from the HTTP request received from Web Agent.
5. Match the rule against the part of HTTP request retrieved in the last step and conclude rule-matching result.
6. Action data is determined based on the matching result and the action declaration in the *RuleMapping*.

7. In terms of the action data concluded in the last step, special flag (i.e. *deny-flag*, *pass-flag*) might be set.

8. Deliver the action data and matching result to Response Module to invoke action.

9. Go to step 1 again until the loop terminates or special flag is set.

The following programming fragment illustrates how this works. In lines 4-5, if a special flag is set true, terminates the rule-mapping loop. Current rule mapping is specified in the lines 8-9. Lines 10-11 select a rule used as attack pattern. Line 13 calls a private method, *dispatchTarget*, to perform pattern matching for each specific "target".

Lines 17-36 show how the private method, *dispatchTarget*, works for GET parameter in HTTP request. The detailed GET parameter data is prepared in line 22. Lines 25-27, rule matching between the specific rule and GET parameter is done, the corresponding analysis result and error comment (it is an empty string, if no intrusion is found) are concluded. Lines 29-30, the action data can be determined through the combination of rule mapping and matching result. Based on the action data, special flag(s) might be set in the line 31. The Response Module is triggered to carry out response actions in line 32.

```
1.  while (ruleMappingContinued) // If ruleMapping continues
2.  {
3.      ...
4.      if (passFlag || denyFlag || redirectFlag) // If passFlag, denyFlag, or redirectFlag is set
5.          break;
6.
7.      // Get current RuleMapping for Pattern Matching
8.      ruleMapping =
9.          currentRuleMapping.getCurrentRuleMapping(ruleConfigData, ruleMappingCount);
10.     String ruleID = ruleMapping.getRuleID(); // Get current RuleID for Pattern Matching
11.     rule = currentRule.getCurrentRule(ruleData, ruleID); // Current Rule for Pattern Matching
12.     // Invoke various RuleMatcher to do Pattern Matching in terms of different target
13.     this.dispatchTarget(rule.getTarget(), rule, actionData);
14.     ...
15. }
16.
17. private void dispatchTarget(String target, RuleObject rule, ActionSet actionData)
18. {
19.     ...
20.     case REQGET: // Target is ReqGet
21.     {
22.         Vector reqGet = currentTarget.getReqGet();
23.         if (reqGet != null)
24.         {
25.             reqGetMatcher.doMatching(reqGet, rule, resultCache); // Do Pattern Matching
26.             boolean matchingResult = reqGetMatcher.getMatchingResult();
27.             String errorComment = reqGetMatcher.getErrorComment();
28.
29.             forwardActionVector=forwardAction.getForwardActionVector(matchingResult,
30.                 ruleMapping); // Get forward Actions and execute
31.             setFlag(forwardActionVector);
32.             actionManager.executeAction(errorComment,forwardActionVector,actionData);
33.         }
34.     }
35.     ...
36. }
```

**Figure 5-2 Programming Details on Rule Matching**

### Special Flags

In this Pattern Matching Engine, four special flags, including deny-flag, pass-flag, redirect-flag and skip-flag, are used to represent the current status of the processing. If one of the four flags is set, the corresponding action will be executed. The detailed explanations can be summarized as follows:

- Deny-flag: when this flag is set, it means that an intrusion is detected. Thus, the incoming HTTP request should be denied.
- Pass-flag: when this flag is true, it indicates that current HTTP request does not trigger any rule in the pattern-matching analysis.
- Redirect-flag: when this flag is true, it implies an error is found. However, client will receive a redirected web page or an error page.
- Skip-flag: when this flag is true, it means a list of rules could be skipped over or ignored without further analysis.

In the Response Module, four response actions corresponding to the four flags are defined. They are deny-action, pass-action, redirect-action and skip-action, respectively. Once an action is triggered, the corresponding flag is set More information about the response action is discussed in the Chapter 6 Response Module.

## 5.3 Behavior Modeling Engine

### 5.3.1 Behavior Modeling Algorithm

The algorithm of Behaviour Modeling is based on anomaly-based detection [2], which is complementary to the misuse detection. In this case, detection is based on models of normal behaviour of users and applications, called "profiles". Any deviations from such established profiles are interpreted as attacks or intrusions. The main advantage of Behaviour Modeling algorithm is that it is able to identify previously unknown attacks. By defining an expected normal state, any abnormal behaviour can be detected, once it cannot fit into the normal behaviour profile.

The Behaviour Modeling algorithm follows a learning-based anomaly detection technique. An example of this technique is described by Forrest [24]. During the training phase, the system collects all distinct system call sequences of a certain specified length. During detection, all actual system call sequences are compared to the set of legitimate ones concluded in the training period, raising an alarm if no match is found.

Therefore, the algorithm works in two modes, learning mode and detection mode. It can be illustrated by the following figure.

- Learning mode: during learning period, models (or profiles) that characterize the normal behaviour of the Web application are built based on the network events observed.
- Detection mode: when Behaviour Modeling Engine works in the detection mode, all incoming requests are compared to the profiles that were established in the learning mode. If an intrusion (the request is deviated from the normal profile) is detected, a related alert can be raised.

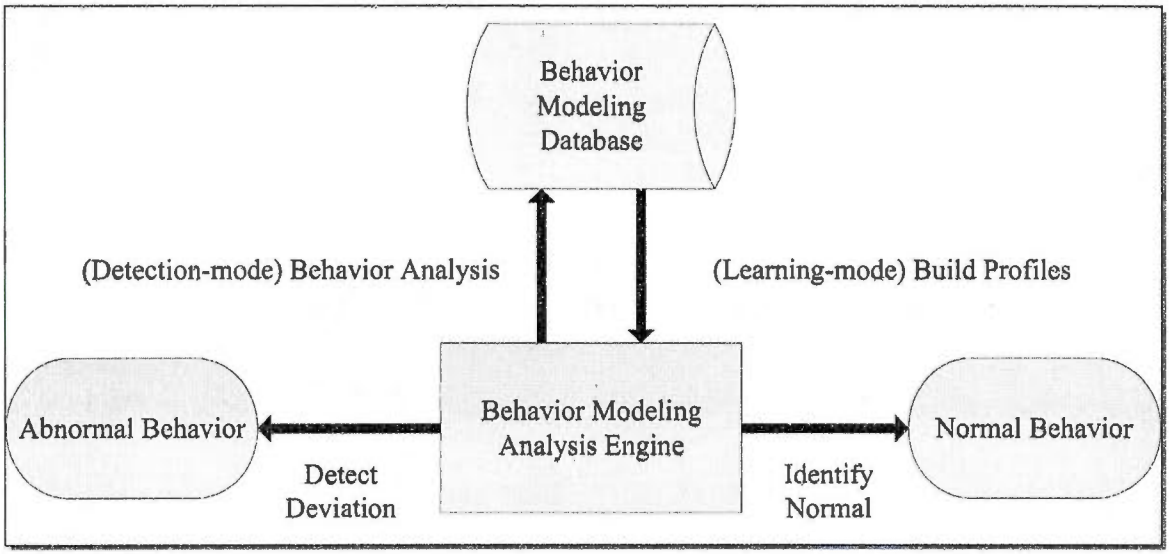


**Figure 5-3 Behavior Modeling Algorithm**

The Behaviour Modeling algorithm conforms to the statistical approach of anomaly detection. In the learning mode, behaviour profiles for subjects (i.e., Request parameter and URI) are generated. A user definable threshold is used to evaluate if the current behaviour could be stored to database as a legitimate profile. After the learning phase, the system switches to detection mode in which the new traffic is compared to the profile for detection of anomalies.

However, since a system can evolve over time, it is also likely that new non-malicious inputs will be seen [3] [25]. In the other word, the false positive rate will increase dramatically after a long time running. It is simple to update the profile-base by the learning phase on the changed traffic when the number of false positive alarms is greater than the pre-defined value.

In the Behaviour Modeling Engine, two Behaviour Modeling algorithms are extended to detect security-related issues in the request URI, and request parameter. We will discuss these two algorithms in the Section 5.3.4 and Section 5.3.5, respectively.

**Design of Behavior Modeling Engine**

- Like Pattern Matching Engine, Behavior Modeling Engine also has to initialize the configuration files before analysis starts. These XML configuration files include model-mapping file and the model definition file.
- Behavior Modeling Engine determines its working modes based on the configuration provided by model-mapping file and model definition file.
- Suppose Behavior Modeling Engine is in learning mode, characterize normal behaviors of events and establish profile.
- If Behavior Modeling Engine is in detection mode, match the incoming requests with legitimate profile. Whenever an analysis result is concluded, Behavior Modeling Engine fires Response Module to execute the corresponding actions.

### 5.3.2 File Description

Like Pattern Matching Engine, initialization should be done before behaviour-modeling analysis starts. Two files are defined here, and they are the model definition file, and model-mapping file.

#### Model File

In the Behaviour Modeling Engine, two modeling algorithms are developed to check request parameter and request URI in the HTTP request. The two modeling algorithms correspond to two "Model" elements declared in the model definition file. Thus, when we build model for request parameter or request URI, the corresponding part of HTTP request will be evaluated according to the definitions in the model definition file. The file structure is illustrated by the following table:

| Item Name | Description |
|---|---|
| ModelID | ID of model |
| ModelName | Name of Model |
| Threshold | An integer value that determines if the activity should be classified as a normal behavior. When the number of times that a behavior occurs is greater than the defined threshold, we would treat it as normal behavior. |
| CurrentMode | Working state of current model; it could be learning or detection mode. |

**Table 5-4 Structure of a Model**

### Model-Mapping File

Just like the rule-mapping file employed in the Pattern Matching Engine, a model-mapping file is defined in the Behavior Modeling Engine. The primary function of this file is to build mapping among application, agent, model and related actions. In the other word, the model-mapping file is a map to demonstrate relationships among models, actions, Web applications and Web Agents.

| Item Name | | Description |
|---|---|---|
| ApplicationID | | ID of Application |
| AgentID | | ID of Web Agent |
| ModelID | | ID of Model |
| forward | name | Two choices for this attribute of *forward* element:<br>match: HTTP request fits the profile determined by *ModelID*<br>unmatch: HTTP request does not fit the profile determined by *ModelID* |
| | actionID | ID of action to be executed. |
| | actionParam | The customized action parameter |

**Table 5-5 Structure of a Model-Mapping**

### The Relationships of Two Files

The following figure illustrates the relationships of the two configuration files, model file and model-mapping file.

- A *ModelID* is specified in the model-mapping file.
- Based on the reference to *ModelID*, the detailed model data can be retrieved from the model definition file.

- The *ModelName* declared in the model definition file determines which part of HTTP information must be examined.
- The *CurrentMode* specified in the model definition file determines the current running mode of the model, learning or detection.
- If "learning mode" is signed, the *threshold* declared in the model definition file provides a lower boundary of the number of times a normal behaviour occurs.
- If "detection mode" is configured, the corresponding action data could be retrieved from the model-mapping file once the analysis result is concluded.

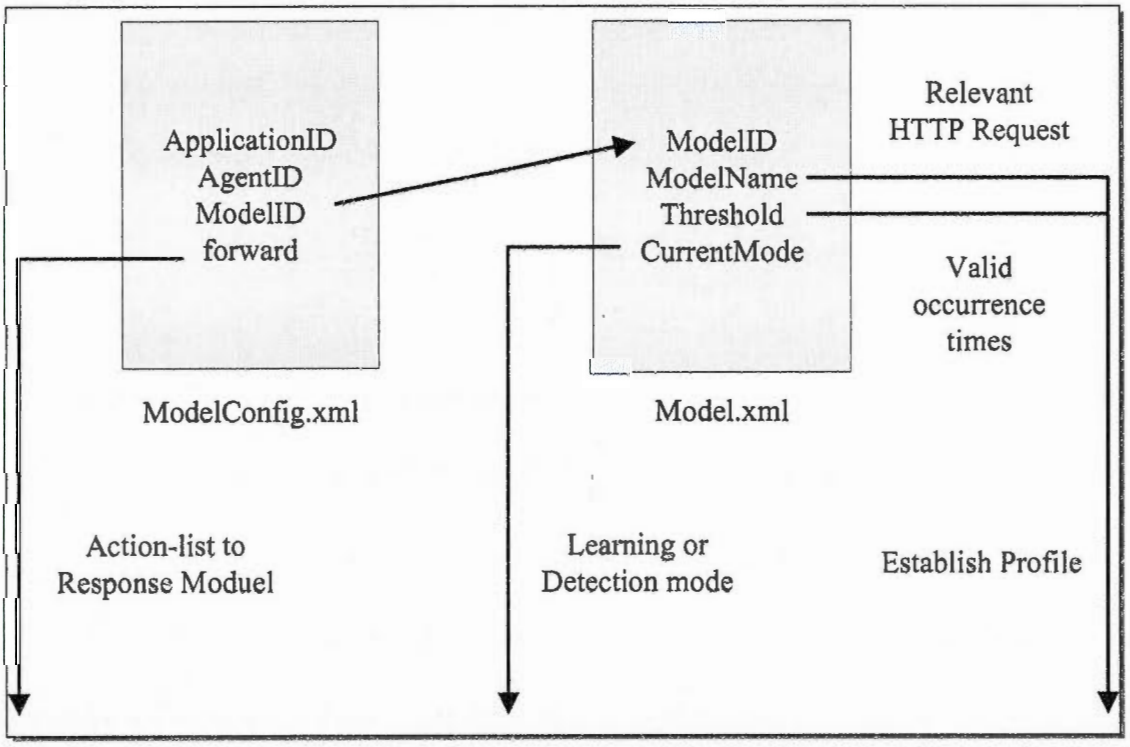


**Figure 5-4 Relationship of Tables in Behavior Modeling Engine**

### 5.3.3 How to Perform Behavior Modeling

#### Basic Algorithm

For each specific Web application and Web Agent, several different *ModelMappings* might be processed. The *ModelMapping* traversal involves following steps:

1. If there is un-visited *ModelMapping* left or special flag (i.e. *deny-flag*, *pass-flag*) is not set, the loop continues; otherwise it terminates.
2. Retrieve current *ModelMapping* from model-mapping file, and the model to be used can be determined subsequently.
3. Once the model is specified, model name is used to refer to the actual model data defined in model definition file.
4. According to the model name specified in the step 3, the detailed HTTP section could be retrieved from the HTTP request forwarded by Web Agent.
5. The corresponding working mode can be determined once a specific model is concluded in the step 3.

**If the model is in learning mode:**

6. If it is in learning mode, invoke learning procedure, establish profile, and go to step 1.

**If the model is in detection mode:**

6. If it is in detection mode, invoke detection procedure and conclude analysis result.
7. Action data can be determined through the combination of matching result and *ModelMapping*.
8. According to the action data deduced in the last step, special flag (i.e. *deny-flag*, *pass-flag*) might be set.
9. Deliver the action data and matching result to Response Module to invoke action.
10. Go to step 1 again until the loop terminates or a special flag is set.

### Special Flags

The special flags, including deny-flag, pass-flag, and redirect-flag, are used in the Behaviour Modeling Engine again. If one of the three flags is set, the corresponding action will be executed. The definitions of the three special flags are same as those in the Pattern Matching Engine. The three response actions corresponding to the special flags are discussed in the Chapter 6 Response Module.

### 5.3.4 URI Modeling Algorithm

The algorithm is designed to verify any abnormal behaviour in the request URI. Exploitations, like Directory Traversal, and other invalid URI requests.

#### Data Structure

Two issues have to be considered when designing data structure used in the learning mode. First, the class for URI modeling defines a "counter" field to count how many times the URI is requested. Thus, a URI address attribute and an attribute for counter are defined in the class. Second, all URIs should compose of a URI collection. So, an instance variable of *List* is constructed to store the all request URI objects. The following figure shows the declaration of a request URI object, *URIStatistic*.

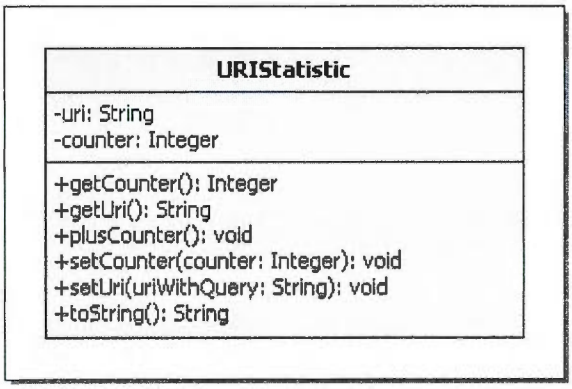


**Figure 5-5 Data Structure of Class URIStatistic**

### Algorithm

- Learning mode: During the learning period, all distinct URIs in the Web application could be learned. For each request URI, we record how many times the URI is requested. When the number is greater than the *threshold* defined in the model definition file, the URI could be stored in database as a URI profile.
- Detection mode: In the detection mode, it matches current request URI with the valid URI profile and returns analysis result.

## 5.3.5 Parameter Modeling Algorithm

The modeling algorithm is used to detect deviation of request parameters, including POST parameters and GET parameters. It can help Web application to prevent from attacks or intrusions related to request parameters, such as SQL Injection or XSS (Cross-Site Scripting), etc.

### Data Structure

We pre-define data types of HTTP request parameter in the database. So, we can determine the type of a request parameter by matching the request parameter with pre-defined parameter type during learning period. Since there are several alternative parameter types for a single parameter, we can design more than one counter to record how many times that a parameter is of a specific type.

In the other word, a request parameter might associate with several counters, and each counter corresponds to a possible parameter type. Thus, we declare two attributes in the *ParamStatistic* class; one is parameter name, the other is a list of counters.

Each counter should also be a compound data, or an object. It is composed of two attributes; one attribute is to mark the identity of possible parameter type, the other is just an integer counter. The following figure shows the data structure of *ParamStatistic* and *ParamTypeCounter*, where *ParamTypeCounter* is an inner class of class *ParamStatistic*.

Moreover, in order to build profiles for all request parameters in the learning mode, the parameters should compose of a parameter collection. So, an instance variable of *List* is constructed to store the all request parameter objects.

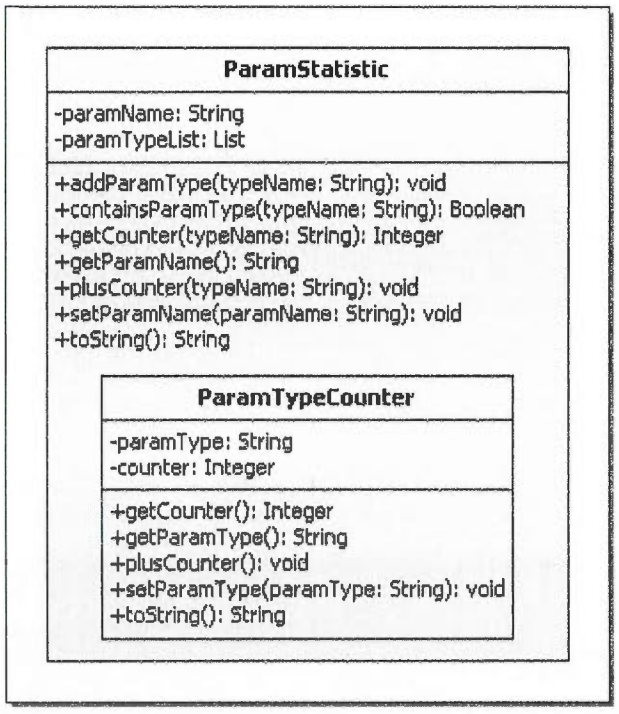


**Figure 5-6 Data Structure of Class ParamStatistic**

**Algorithm**

- Learning Mode

  In the learning mode of parameter modeling, all parameters of the Web application have to be profiled. We have all possible parameter types stored in a table in the database. And these parameter types are described with Regular Expression. For each request parameter, we record how many times this parameter matches to a specific parameter type. When the number exceeds the pre-defined threshold declared in the model definition file, the parameter name and data type are inserted into database as a legitimate profile.

- Detection Mode

  During detection period, it will verify if the current request parameter matches to legitimate parameter profile derived in the learning mode, and returns the analysis result.

## 5.4 Conclusion

Analysis Engine module is the most important part in the Intrusion Detection framework for Web application. The responsibility of Analysis Engine is to receive captured HTTP request from Web Agent and detect if there is any intrusion or malicious exploitation in the request. Then, Analysis Engine triggers Response Module to carry out specific actions based on the analysis result. In this chapter, detection algorithms including Pattern Matching based on Misuse algorithm and Behaviour Modeling based on Anomaly-based detection are discussed.

# CHAPTER 6

# RESPONSE MODULE

The focus of this chapter is to reveal the implementation details associated with Response Module. Firstly, we introduce a configuration file. It is used to define actions deployed in the Response Module. Then, implementation of Response Module is discussed.

## 6.1 Description

According to analysis result and pre-defined actions, Response Module takes corresponding actions. Moreover, it sends action instruction to Web Agent.

If an intrusion is detected, "deny" action must be triggered, an alert has to be sent to administrator of the Web application. Additionally, malicious HTTP request data will be inserted into database. As a result, Web Agent will receive an analysis result action instruction, and the malicious attacker will get an error page or request-forbidden page.

If no intrusion exists, Response Module will inform the web Agent that the current request is legitimate, and the client will get his or her requested page.

**Design of Response Module**

- The configuration file, action definition file, must be initialized in advance before Response Module is triggered.
- Based on the analysis result and action data sent from Analysis Engine, Response Module executes corresponding actions.
- Returns action instruction(s) to Web Agent. Then, Web Agent can respond different pages to clients.

## 6.2 File Initialization

Action definition file is the configuration file for Response Module to declare the actions that will be executed. The actions will be fired once the analysis result of signature detection or anomaly detection is concluded. Table 6-1 shows the structure of an action.

| Item Name | Description |
|---|---|
| ActionID | ID of action |
| ActionCom | Command for this action |
| ActionComParam | Some parameters related with this action. Take skip action for example, a list of rules can be defined here which indicate the given rules can be skipped over during further Pattern Matching Analysis. |

**Table 6-1 Structure of an Action**

### Actions in Response Module

There are seven unique actions defined in the Response Module, they are *deny action*, *pass action*, *continue action*, *skip action*, *redirect action*, *exec action*, and *logcontent action*. The detailed explanations of these actions are as follows:

- *Deny action:*

  When this action is triggered, it indicates that an intrusion is detected and the current HTTP request should be denied. Customized message can be defined in the *actionComParam* to describe the reason that HTTP request is denied. Even if there is other rules or models left or un-visited in the analysis loop, the current analysis terminates immediately.

- *Pass action:*

  If current incoming HTTP request passes the analysis of pattern matching or behavior modeling, this action will be performed. When this action is fired, current analysis terminates even if there are other rules or models left or un-visited in the analysis loop.

- *Continue action:*

  Unlike *Pass action*, when *Continue action* is fired, next rule or model in sequence will be executed continually instead of ignoring them.

- *Skip action:*

  When this action is triggered, a number of rules that specified in the *actionComParam* element will be ignored.

- *Redirect action:*

  If an intrusion or attack is detected, *Redirect action* could be alternative choice. This redirected page is the customized web page specified by administrator. A redirected URL could be specified in the *actionComParam* element.

- *Exec action:*

  Administrator can declare some external commands in the *actionComParam* element of this action. For example, send an alert or alarm to administrator by email when an intrusion is confirmed.

- *Logcontent action:*

  When an intrusion is detected, intrusion log should be generated. Thus, the job of this action is to produce log. The open source software Log4j is deployed to manage the logging conveniently.

## 6.3 Implementation of Response Module

### 6.3.1 Execution of Response Action

According to the definitions in the model-mapping or rule-mapping file, the action data delivered from Analysis Engine includes *actionID* and *actionComParam*. With the combination of the received action data and action definition file, Response Module executes specified action(s). The equivalent pseudo code is stated below.

**Pseudo Code**

1. Argument *forwardedActions* indicates the action data sent from Analysis Engine. Only *actionID* and *actionComParam* are included in it.
2. Argument *analysisResult* represents the matching result set by the Analysis Engine. It includes intrusion description or error message.
3. We use *actionID[forwardAction]* and *actionParam[forwardAction]* to hold *actionID* and action command parameter of a response action, respectively.
4. We use *actionID[action]* and *actionCom[action]* to hold *actionID* and action command of a pre-defined action declared in the action definition file.

The algorithm *ResponseModule* works as follows. The *for* loop of lines 1-7 execute each action determined by *actionID*. Lines 2-3 get *actionID* and action parameter of current action to be fired. Line 4 calls procedure *ResponseModule-Get-ActionCom* to retrieve action command corresponding to the given *actionID*. Lines 5-6 handle the case in which the returned action comment is valid. Line 6 carries the determined action into execution. Line 7 prints an error message when action command is null.

```
ResponseModule (forwardedActions, analysisResult)
1.   for each forwardAction in the forwardActions
2.       currentActionID ← actionID[forwardAction]
3.       currentActionComParam ← actionParam[forwardAction]
4.       currentActionCom ← ResponseModule-Get-ActionCom (currentActionID)
5.       if currentActionCom ≠ NIL
6.          execute action specified by currentActionCom with given currentActionComParam
7.       else error "Invalid ActionID"
8.   return

ResponseModule-Get-ActionCom (currentActionID)
1.   actions ← Actions declared in the action definition file
2.   currentActionCom ← NIL
```

**Figure 6-1 Pseudo Code for Response Module**

### 6.3.2 Send Action Instruction to Web Agent

An action instruction should be returned to Web Agent, no matter whether an intrusion is detected in the HTTP request or not. With the result from Analysis Engine, Web Agent can return different web pages to client. If an intrusion is recognized, a client might receive an error page or redirected page; otherwise, he or she would get requested page.

A class named *ResponseActionServlet* that extends HttpServlet is developed to send response message from WebIDS to Web Agent.

- When an intrusion is detected, the malicious HTTP request must be inserted into database for further analysis or kept as intrusion evidences. And a short error description will be returned to Web Agent.
- If no intrusion is detected, "OK" respond should be sent to Web Agent.

## 6.4 Conclusion

In this chapter, the implementation of Response Module is discussed. The primary function of this component is to execute response actions according to matching result and action data from the Analysis Engine. Moreover, the process for making response to Web Agent is also discussed.

# CHAPTER 7

# DATABASE

In chapter 7, WebIDS database design is reviewed. Tables used to store malicious HTTP request and tables used in the Behavior Modeling Engine to keep established profiles are reviewed. Finally, we discuss on the data access technology in WebIDS supported by Spring-framework and Hibernate briefly. Two databases are created in this project. One is for the storage of malicious HTTP requests, and the other is to store the configuration for behavior-modeling analysis.

## 7.1 Database for HTTP Request

If an intrusion or attack is detected, the malicious HTTP request must be saved to database for further analysis. Therefore, a database named *HttpInfoDB* is created to hold those data. Since the HTTP requests sent from Web Agent are broken down into seven parts, and seven tables are created for them correspondingly.

They are Web application table, Web Agent table, client table, HTTP header table, session table, cookie table, and request parameter table. The tables are designed as followings:

- Application Table:

| Field Name | Description |
|---|---|
| ID | ID of application information (Primary Key) |
| ClientID | ID of client (Foreign Key) |
| AppName | Name of Web application |
| AppIP | IP address of Web application |
| AppPort | Port of Web application |

**Table 7-1 Structure of Malicious HTTP Request: Application**

- Agent Table:

| Field Name | Description |
|---|---|
| ID | ID of Web Agent record (Primary Key) |
| ClientID | ID of client (Foreign Key) |
| AgentName | Name of Web Agent |

**Table 7-2 Structure of Malicious HTTP Request: Agent**

- Client Table:

| Field Name | Description |
|---|---|
| ID | ID of client record (Primary Key) |
| ClientName | Name of client machine |
| ClientIP | IP address of client machine |
| ClientPort | Port of client machine |

**Table 7-3 Structure of Malicious HTTP Request: Client**

- HTTP Header Table:

| Field Name | Description |
|---|---|
| ID | ID of header record (Primary Key) |
| ClientID | ID of client (Foreign Key) |
| HeaderName | Name of HTTP header item |
| HeaderValue | Value of HTTP header item |

**Table 7-4 Structure of Malicious HTTP Request: Header**

- Session Table:

| Field Name | Description |
|---|---|
| ID | ID of session record (Primary Key) |
| ClientID | ID of client (Foreign Key) |
| SessionID | Identifier that the servlet container assigns to the session |
| SessionCreated | The time when the session is created |
| SessionLastAccessed | The last time that a client sends a request associated with the session |

**Table 7-5 Structure of Malicious HTTP Request: Session**

- Cookie Table:

| Field Name | Description |
|---|---|
| ID | ID of cookie record (Primary Key) |
| ClientID | ID of client (Foreign Key) |
| CookieKey | Name of the cookie |
| CookieValue | Value of cookie |

**Table 7-6 Structure of Malicious HTTP Request: Cookie**

- Request Parameter Table:

| Field Name | Description |
|---|---|
| ID | ID of parameter record (Primary Key) |
| ClientID | ID of client (Foreign Key) |
| Method | Method of HTTP request, GET or POST |
| ParamKey | Name of request parameter |
| ParamValue | Value of request parameter |

**Table 7-7 Structure of Malicious HTTP Request: Parameter**

## 7.2 Database for Behavior Modeling Engine

A database named *BehaviorModelingDB* is designed to facilitate the Behavior Modeling analysis. There are static tables as well as dynamic tables used in the Behavior Modeling Engine. Static table is created before behavior-modeling analysis begins. Dynamic table stores the profiles established during learning phase.

### The Static Table

In the parameter modeling, a static table stores pre-defined parameter in the database. The structure of table *ParamTypeInfo*, can be illustrated as follow:

| Field Name | Description |
|---|---|
| ID | ID of parameter type (Primary Key) |
| TypeName | Name of the parameter type |
| ParamType | Parameter type pattern described with Regular Expression. During learning period, it might be retrieved to match against each request parameter. |

**Table 7-8 Structure of ParamTypeInfo**

### The Dynamic Table

The dynamic tables keep web application profiles generated during learning mode of behavior-modeling analysis. There are two dynamic tables *ParamProfile* and *URIProfile* in the *BehaviorModelingDB*. They can be illustrated by the following figures, respectively:

- ParamProfile Table:

| Field Name | Description |
|---|---|
| ID | ID of parameter profile record (Primary Key) |
| ParamName | Parameter name |
| TypeID | ID of parameter type (Foreign Key). It determines parameter type of a given parameter concluded in the learning phase. |

**Table 7-9 Structure of ParamProfile**

- URIProfile Table:

| Field Name | Description |
|---|---|
| ID | ID of URI profile record (Primary Key) |
| URI | The URI profile value. Any deviated URI can be specified in comparison with this URI during detection period. |

**Table 7-10 Structure of URIProfile**

## 7.3 Data Access Supported by Spring and Hibernate

The hierarchy of data access under the support of Spring and Hibernate includes Logic Representation Layer, Business Service Layer, and Persistent Object Layer.

- Persistent Object Layer:

  Persistence Layer is under the management of Hibernate [8], where some persistent classes are created. The XML files required for the object/relational mapping are declared here too.

- Business Service Layer:

  Spring [23] manages Business Layer, Spring's lightweight bean container offers IoC-style (Inversion of Control) wiring up of business objects, DAOs, and resource like JDBC DataSources and Hibernate SessionFactories. The following figure shows the definition Business Service Layer for database, *BehaviorModelingDB*.

- Logical Presentation Layer:

  Logical Presentation Layer makes use of business service provided by the Business Layer to implement logical consideration when database-related operations are requested. For example, when we intend to insert legitimate parameter profile or URI profile in the learning mode of the Behaviour Modeling, we just create a new persistent object and request insertion service provided by the Business Layer.

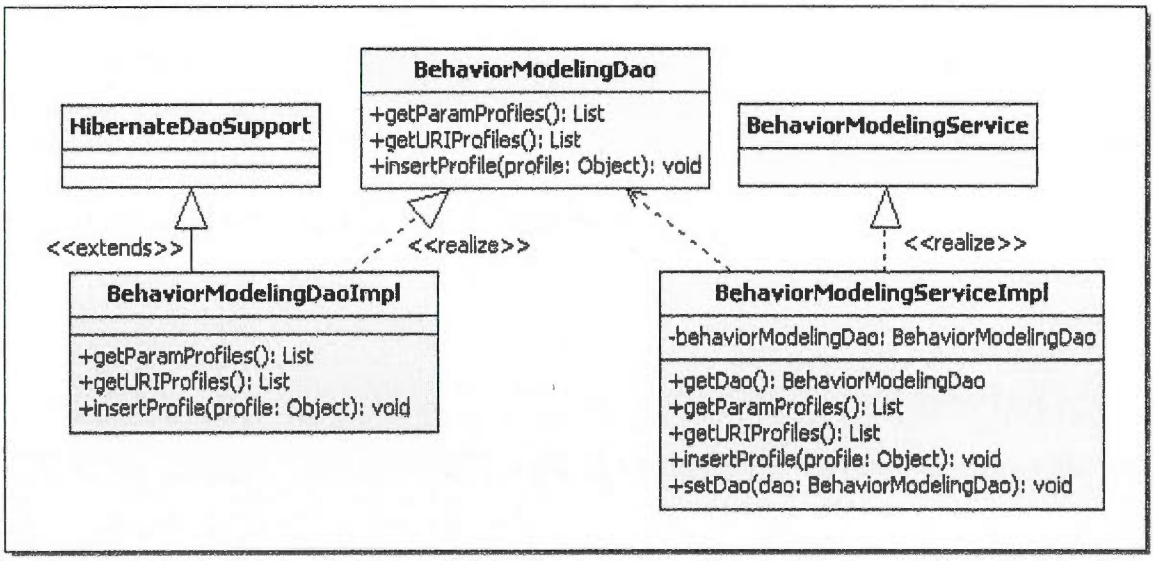


**Figure 7-1 Business Service Layer of BehaviorModelingDB**

## 7.4 Conclusion

The primary function of database is to provide persistent storage for the entire Intrusion Detection System framework. In this chapter, databases and tables are discussed, including tables for malicious HTTP request and tables used to keep profiles in the Behaviour Modeling Engine. The data access under the support of Spring-framework and Hibernate makes database-related operations more straightforward.

# CHAPTER 8

# EXPERIMENTAL RESULTS AND CASE STUDIES

In this chapter, functionality testing on the Web Application intrusion detection framework is discussed. Some common intrusion test cases are performed, including SQL Injection, Cross-site scripting, Directory Traversal and Hidden Field. These simulations are used to demonstrate efficiency and effectiveness of the proposed framework.

The simulation is done on a workstation with 2.4GHZ CPU and 512MB DDR running Windows© 2000 Server under light load. The web server is Tomcat 5.5.2 [28]. The proposed WebIDS is executed in the same time along with the simulating HTTP requests.

## 8.1 Case Studies

We tested this framework with several attacking test cases. The intrusion detection framework can effectively detect the attacks and raise alarms when exploitations occur.

### 8.1.1 SQL Injection

#### Description

According to the Top Ten Most Critical Web Application Security Vulnerabilities [29] issued by OWASP [16] (Open Web Application Security Project), Injection Flaws is the 6th of the top ten most critical vulnerabilities.

SQL Injection is one kind of widespread and dangerous Injection Flaws. To exploit a SQL Injection flaw, the attacker must find a parameter that the web application uses to dynamically construct a SQL query. *"By carefully embedding malicious SQL commands into the content of the parameter, that attacker can trick the Web application into forwarding a malicious query to the database [11]."*

#### Simulation

Web Agent captures HTTP request, and normalizes them into XML format. The following figure illustrates the collected malicious parameter to exploit SQL Injection flaw:

```
<Request>
  <RequestParam>
    <Name>account_number</Name>
    <Value>' or '1=1</Value>
  </RequestParam>
</Request>
```

**Figure 8-1 Request Parameter for SQL Injection**

Pattern-matching analysis can utilize pattern to detect SQL Injection intrusion. The rule is illustrated with Figure 8-2:

```
<SysRule>
  <ID>Rule032</ID>
  <RuleName>SQLInjection(1=1)</RuleName>
  <Version>1.0</Version>
  <Target>ReqPost</Target>
  <Pattern>[0-9]{1,}\W{0,}=\W{0,}[0-9]{1,}</Pattern>
  <CaseSensitive>y</CaseSensitive>
  <Length>128</Length>
  <ReferenceNo>no</ReferenceNo>
</SysRule>
```

**Figure 8-2 Rule for SQL Injection**

The intrusion log demonstrating the occurrence of SQL Injection Exploitation looks like:

```
2005-03-17 20:42:20,250 - Alert: ClientIP: 127.0.0.1 ClientPort: 1334  Intrusion Attempt detected: SQLInjection
```

**Figure 8-3 Intrusion Log for SQL Injection**

The attacker or a malicious client might get following error page:

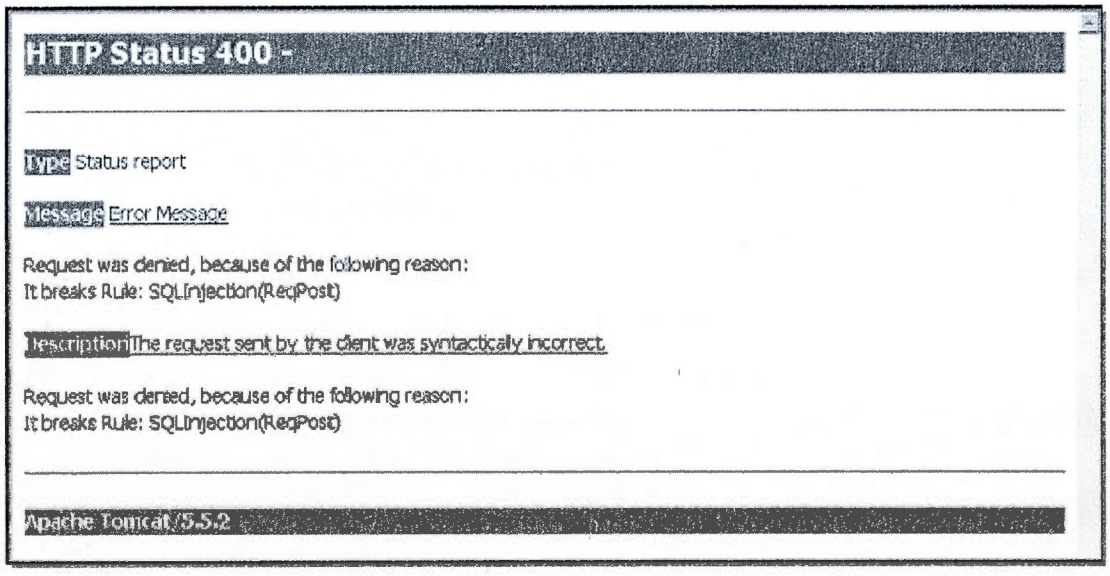


**Figure 8-4 Error Page for SQL Injection**

### 8.1.2 Cross-Site Scripting (XSS)

The Cross-Site Scripting attack is also an injection flaw, which is one of the most critical and common Web application vulnerabilities defined by OWASP [16]. These flaws occur when an attacker uses a Web application to send malicious code, generally in the form of a script, to different end user [6]. These vulnerabilities are quite widespread and occur when a Web application takes user input as output directly without validating it.

Pattern-matching analysis is very efficient to detect Cross Site Script attack. Therefore, a rule to detect this type of attack is described in the following table:

```
<Request>
  <RequestParam>
    <Name>msg</Name>
    <Value><SCRIPT>alert(document.cookie);</SCRIPT></Value>
  </RequestParam>
</Request>
```

**Figure 8-5 Request Parameter for XSS**

```
<SysRule>
  <ID>Rule026</ID>
  <RuleName>XSS(ReqPost)</RuleName>
  <Version>1.0</Version>
  <Target>ReqPost</Target>
  <Pattern>(&lt;script&gt;|\.cookie)</Pattern>
  <CaseSensitive>n</CaseSensitive>
  <Length>128</Length>
  <ReferenceNo>no</ReferenceNo>
</SysRule>
```

**Figure 8-6 Rule for XSS**

```
2005-04-10 20:42:16,828 - Alert: ClientIP:127.0.0.1 ClientPort:1334 Intrusion Attempt detected: XSS
```

**Figure 8-7 Intrusion Log for XSS**

### 8.1.3 Directory Traversal

Directory Traversal is another attack that breaks Web application input validation. *"It is an attempt to access files outside of the Web document root, or files within the document root, which are otherwise restricted to the user [1]."* The primary target of directory traversal attack is the URL an attacker can manipulate to bypass the system access control.

Both Pattern Matching Engine and URI modeling of Behavior Modeling Engine are able to detect Directory Traversal attack. For a possible directory traversal intrusion, the corresponding pattern-matching rule and intrusion log are illustrated as follows:

```
<Request>
  <RequestParam>
    <Name>File</Name>
    <Value>%2E%2E%2F%2E%2E%2F</Value>
  </RequestParam>
</Request>
```

**Figure 8-8 Request Parameter for Directory Traversal**

```
<SysRule>
  <ID>Rule016</ID>
  <RuleName>DirectoryTraversal(ReqPost)</RuleName>
  <Version>1.0</Version>
  <Target>ReqURI</Target>
  <Pattern>\.\./</Pattern>
  <CaseSensitive>n</CaseSensitive>
  <Length>128</Length>
  <ReferenceNo>no</ReferenceNo>
</SysRule>
```

**Figure 8-9 Rule for Directory Traversal**

```
2005-04-10 20:42:20,250 - Alert: ClientIP:127.0.0.1 ClientPort:1334 Intrusion Attempt detected: DirectoryTraversal
```

**Figure 8-10 Intrusion Log for Directory Traversal**

### 8.1.4 Hidden Field

#### Description

Behaviour Modeling Engine can detect Directory Traversal efficiently. Cross-Site Scripting and SQL Injection Flaws can be detected if Parameter profile is built correctly. However, only Behaviour Modeling Engine can stop Hidden Field manipulation attack effectively.

*"HTML can store field values as Hidden Fields, which are not rendered to the screen by the browser but collected and submitted as parameters during submissions [1]."* The attacker can save the source code for this HTML page, change the hidden field value or change its value by Proxy tools during the submission, and then post the newly changed value to the Web application. This attack is dependent on the deviation from normal parameter value. Thus, only Behaviour Modeling Engine can raise an alarm when this exploitation occurs.

#### Simulation

During the learning mode, a parameter profile to describe the normal behaviour of this hidden field, *Price*, can be established. Following table illustrate the legitimate profile in the database.

| TypeID | TypeName | ParamType |
|---|---|---|
| 1 | Hidden | 4999.99 |

**Table 8-1 Parameter Type for Hidden Field**

| ParamID | ParamName | TypeID |
|---|---|---|
| 20 | Price | 1 |

**Table 8-2 Parameter Profile for Hidden Field**

During the detection mode, we use the parameter name of this hidden field, *price*, as a key to retrieve its legitimate parameter type. Then, the result 4999.99 can be gotten and a pattern represented with regular expression can be created consequently. Thus, the deviation can be found by matching current malicious value with the legitimate profile.

And now an intrusion attempts to manipulate this hidden field is demonstrated with of a Proxy tool, Paros [17]. The following figure illustrates this attack:

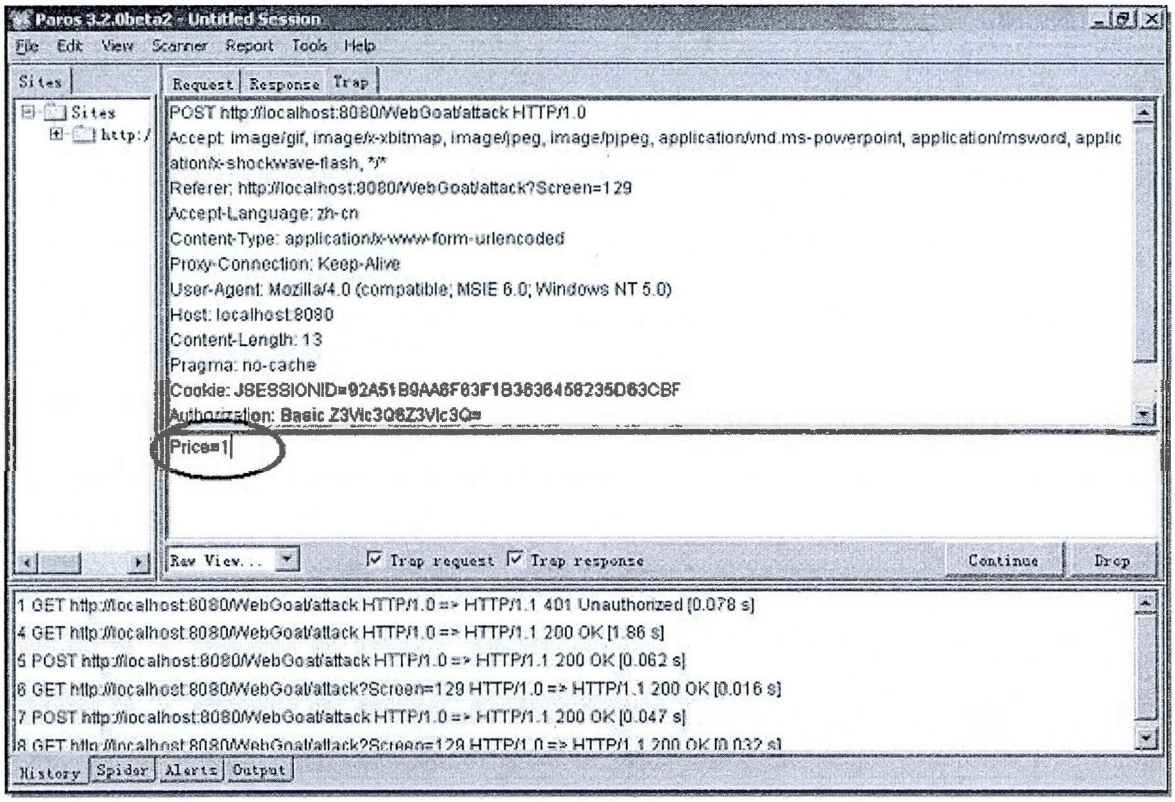


**Figure 8-11 Manipulation of Hidden Field with Paros**

The following figures illustrate the generated intrusion log and the error page that Web Agent returns to an attacker:

```
2005-04-10 20:42:25,625 - Alert: ClientIP:127.0.0.1 ClientPort:1334 Intrusion Attempt detected: Invalid parameter type: [Price = 1]
```

**Figure 8-12 Intrusion Log for Hidden Field**

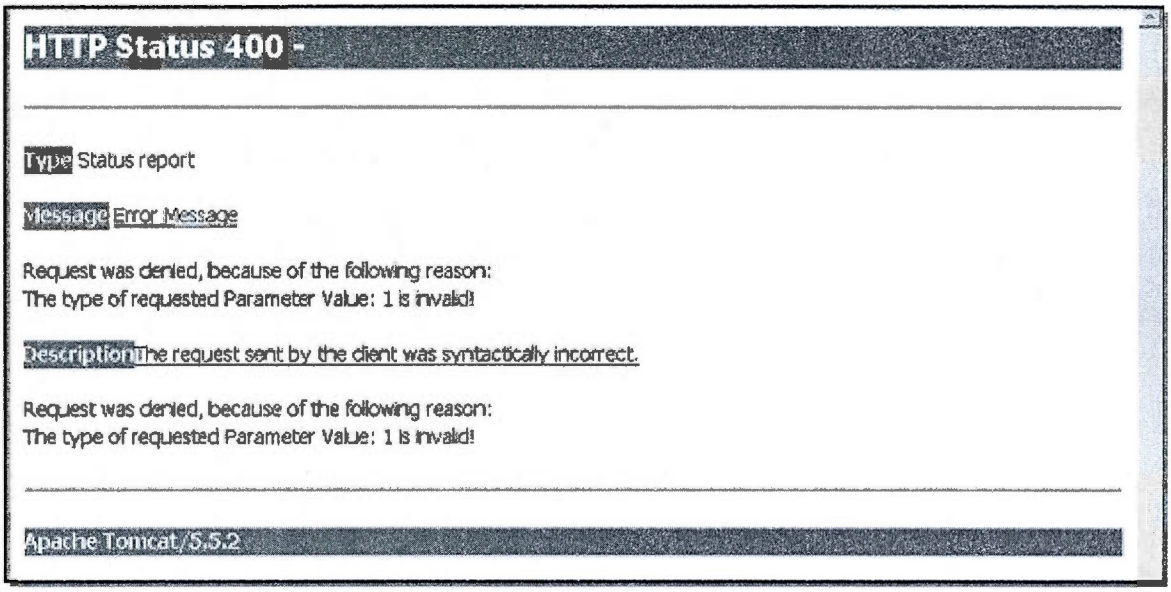


**Figure 8-13 Error Page for Hidden Field**

## 8.2 Conclusion

In this chapter, intrusion test cases to evaluate the effectiveness of this system are discussed. These simulations demonstrate that the Intrusion Detection Framework for Web application proposed in the thesis can protect Web application against malicious attacks efficiently and effectively.

# CHAPTER 9

# CONCLUSION AND RECOMMENDATION

In this chapter, the conclusion on this project is summarized. Moreover, some recommendations or considerations for further research are also discussed here briefly.

## 9.1 Conclusion of Thesis

In this thesis, an Intrusion Detection framework for Web application with advanced detection algorithms, which is based on module architecture and compatible with J2EE development standard, is proposed.

Advanced intrusion detection algorithms, including Pattern Matching and Behavior Modeling, are adopted and extended. In the Pattern Matching Engine, new patterns can be easily developed with the XML-based configuration files. Behavior Modeling Engine can be easily configured and extended. A learning-based anomaly detection algorithm, Behavior Modeling, makes the detection of unknown attack possible. These methodologies and mechanisms help this framework detect the complex as well as simple intrusion attempts efficiently and effectively.

A filter based Web Agent is designed to be a pluggable embedded component within Web application. This allows us to easily deploy WebIDS agent into Web application without additional configuration.

This thesis conducts experiments to evaluate this intrusion detection framework. These experiments successfully demonstrate that this framework can detect various intrusions efficiently and effectively.

## 9.2 Recommendation

Some recommendations for the future works can be summarized as follows:

- For Pattern Matching Analysis Engine, more rules or patterns have to be developed in order to detect more intrusion incidents successfully.
- For Behavior Modeling Analysis Engine, more profiles used to describe HTTP request behavior should be considered.
- For Behavior Modeling Analysis Engine, a more complex and advanced algorithm could be developed to perform statistics analysis. So the statistics analysis can be used to measure whether current request is normal behavior or not more precisely.

- For Behavior Modeling Analysis Engine, more attributes of normal behavior could be tested to model the normal behavior more precisely. The more accurately that normal behavior can be modeled, the less false positive or negative alarms will rise. More experiments should be done to determine the proper attributes to model the behavior.
- For Analysis Engine, a correlation algorithm used to measure the correctness of intrusion detection result should be considered.
- Other type of Web Agents could be developed to collect HTTP request data from other source, such as network Sniffer Agent.

# BIBLIOGRAPHY


1. A Guide to Building Secure Web Applications, Open Web Application Security Project. URL: http://www.owasp.org/documentation/guide.html.
2. A. K. Ghosh, J. Wanken, and F. Charron. Detecting Anomalous and Unknown Intrusions against Programs. In *Proceedings of the Annual Computer Security Application Conference (ACSAC'98), pages 259—267,* Scottsdale, AZ, December 1998.
3. A. Somayaji and S. Forrest. Automated Response Using System-Call Delays. *In Proceedings of the 9th USENIX Security Symposium,* August 2000.
4. Christopher Krügel, Thomas Toth and Engin Kirda. Service Specific Anomaly Detection for Network Intrusion Detection. *ACM Symposium on Applied Computing,* 2002.
5. Common Intrusion Detection Framework (CIDF). URL: http://www.isi.edu/gost/cidf/.
6. *Cross-site Scripting Flaws, Top Ten Most Critical Web Application Security Vulnerabilities – 2004 Updates,* page 15-16, Open Web Application Security Project, January 27th, 2004. URL: www.owasp.org/documentation/topten.html.
7. Donn B. Parker. Demonstrating the Elements of Information Security with Threats. In *Proceedings of the 17th National Computer Security Conference,* pages 421-430, 1994.
8. Hibernate – Relational Persistence for Idiomatic Java, version 2.1.6, August 9th, 2004. URL: http://www.hibernate.org/.
9. http://www.webopedia.com/DidYouKnow/Computer_Science/2005/intrusion_detection_prevention.asp.
10. H. Debar, M. Dacier, and A. Wespi, Towards a Taxonomy of Intrusion Detection Systems, Computer Networks, 31(8):805-822, April 1999.
11. Injection Flaws, Top Ten Most Critical Web Application Security Vulnerabilities – 2004 Updates, page 18-19, Open Web Application Security Project, January 27th, 2004. URL: www.owasp.org/documentation/topten.html.
12. International Standards Organization: *Information Processing Systems – Open Systems Interconnection – Basic Reference Model,* part 2: Security Architecture 7498/2.
13. K. Ilgun, R. A. Kemmerer, and P. A. Porras. State transition analysis: A rule-based intrusion detection approach. *IEEE Transactions on Software Engineering,* 21(3):181–199, March 1995.
14. Kyle Gabhart. J2EE Pathfinder: Filtering with Java Serlvet 2.4 Viewing, extracting, and manipulating HTTP data with Servlet filters, http://www-128.ibm.com/developerworks/java/library/j-pj2ee10.html.

15. K. Jackson, Intrusion Detection Systems (IDS): Product Survey, Los Alamos National Laboratory, LA-UR-99-3883, 1999.
16. Open Web Application Security Project (OWASP). URL: http://www.owasp.org.
17. Paros, version 3.2.1-win, available via http://www.parosproxy.org, April 2006.
18. Prelude Hybrid Intrusion Detection System, version 0.9.0, available via http://prelude-ids.org/, September 2005.
19. Marty Roesch. Snort, version 2.4.4, available via http://www.snort.org/, April 2006.
20. Research Firm Gartner, http://www.gartner.com/.
21. Richard Heady, George Luger, Arthur Maccabe, and Mark Servilla. *The architecture of a network level intrusion detection system*. Technical Report CS90-20, Department of Computer Science, University of New Mexico, August 1990.
22. Sandeep Kumar and Eugene H. Spafford. A Pattern Matching Model for Misuse Intrusion Detection. In *Proceedings of 17th National Computer Security Conference*, pages 11-21, October 1994.
23. Spring – Java/J2EE Application Framework, version 1.1.2, November 2004. URL: http://www.springframework.org.
24. S. Forrest. A Sense of Self for UNIX Processes. *In Proceedings of the IEEE Symposium on Security and Privacy*, pages 120-128, OaKland, CA, May 1996.
25. S. Forrest, A. Somayaji, and D. Ackley. Building Diverse Computer Systems. *In Proceedings of the 6th Workshop on Hot Topics in Operating Systems*, pages 67-72, 1997.
26. Terry Escamilla, Intrusion Detection: Network Security beyond the Firewall, John Wiley & Sons, Inc, 1998.
27. The J2EE™ 1.4 Tutorial for Sun Java System Application Server Platform Edition 8 2004Q4 Beta, Filtering Requests and Responses, pages 503-508, Sun Microsystems, Inc., August 30, 2004.
28. Tomcat, version 5.5.2, available via http://tomcat.apache.org.
29. Top Ten Most Critical Web Application Security Vulnerabilities - 2004 Updates, Open Web Application Security Project, January 27th, 2004. URL: www.owasp.org/documentation/topten.html.
30. T. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, P. Neumann, H. Javitz, A. Valdes, and T. Garvey. A real-time intrusion detection expert system (IDES) – final technical report. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, February 1992.
31. UTF-8: Unicode Organization, available at http://www.unicode.org.
32. ISO 10646: Universal multi-octet character set – UCS, available at http://www.iso.org.