Intrusion Detection and Prevention Framework for Java Web Applications Using Aspects and Autonomic Elements

by

Lei Lin

B.Eng, Hefei University of Technology, China, 1996

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

© Lei Lin, 2010
University of Victoria

All rights reserved. This thesis may not be reproduced in whole or in part, by photocopy or other means, without the permission of the author.

Supervisory Committee

Intrusion Detection and Prevention Framework for Java Web Applications Based on Aspects and Autonomic Elements

by

Lei Lin

B.Eng, Hefei University of Technology, China, 1996

Supervisory Committee

Dr. Hausi A. Müller (Department of Computer Science)

Supervisor

Dr. Yvonne Coady (Department of Computer Science)

Departmental Member

Dr. Kin Fun Li (Department of Electrical and Computer Engineering)

Outside Member

Abstract

Supervisory Committee

Dr. Hausi A. Müller (Department of Computer Science)

Supervisor

Dr. Yvonne Coady (Department of Computer Science)

Departmental Member

Dr. Kin Fun Li (Department of Electrical and Computer Engineering)

Outside Member

Web applications have become increasingly popular in recent years. They are widely used in security-critical areas, such as financial, medical, and military systems. Meanwhile, the number and sophistication of attacks against web applications have increased rapidly. It is important for organizations and companies to add security functions to existing web application servers in order to maintain the confidentiality of critical information. One common approach to protect web systems is to build an Intrusion Detection and Prevention System (IDPS).

In this thesis, we propose an IDPS framework to detect and prevent web attacks by employing Aspect-Oriented Programming (AOP) and Autonomic Computing (AC) technologies. This framework can also be used to discover whether a web application under protection has abilities to prevent certain web attacks itself. We developed a prototyping tool to implement the functionality of this framework partially. We evaluated

this tool on two Java web applications to detect and prevent Cross Scripting Site (XSS) and Structured Query Language (SQL) Injection, which are two of the most common web attacks. The experimental results show that the prototyping tool based on AOP and AC technologies can be applied to detect and prevent the two common web attacks effectively.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I would like to give my special appreciation to my supervisor, Dr. Hausi A. Müller, for his guidance, support, and encouragement throughout my graduate studies. He not only provided me with an excellent research opportunity and environment, but also inspired me to extend my knowledge and improve my skills over the past years. It has been an inspiring and worthwhile experience in my life.

I would also like to thank all the members of the Rigi research group for their contributions. In particular, I would like to acknowledge the great help I received from Qin Zhu, Ron Desmarais, Tony Lin, and Feng Zou.

Finally, I would like to thank my family and friends for all their encouragement and support.

Dedication

To My Family

Thanks for your love and encouragement.

Chapter 1 Introduction

"The vulnerability assessments conducted by WebCohort's Application Defense Center (ADC) concluded that at least 92% of web applications are vulnerable to some form of hacker attacks. The most common vulnerabilities were cross-site scripting (80%), SQL injection (62%) and parameter tampering (60%). While these types of hacking attacks are common, most enterprises have not adequately secured web sites, applications and servers against them. Despite common use of defenses such as firewalls and intrusion detection or prevention systems, hackers can access valuable proprietary and customer data, shut-down websites and servers, defraud businesses, and introduce serious legal liability without being stopped or, in many cases, even detected."

—WebCohort, Inc. [20]

1.1 Motivation

Web-based applications are hugely popular and allow individuals, companies and organizations to conduct business and provide services to their clients. With the discovery of vulnerabilities in web applications, the exploitation of security flaws in web applications has grown significantly over the past decade. In 2001, "Gartner Group reported that 75% of cyber attacks and Internet security violations are generated through Internet Applications [20]." In 2007, the Open Web Application Security Project (OWASP) issued the top 10 serious web application vulnerabilities such as Cross Scripting Site (XSS) and SQL Injection [17]. Even with the rapid development of Internet technologies, web applications have not achieved the desired security level. As a result, web servers and web-based applications are popular attack targets [18]. However, hackers are not satisfied with entering computer systems and controlling them silently.

They can also steal resources from a system using scripting languages, planting malicious code into a normal data stream such as SQL injection, redirecting the web address of the website to a malevolent website, or entering similar webpage resources to read user information directly.

The security of sensitive information that could be retrieved through web access is an increasing concern for organizations as well as for the individuals. One of the top priorities for computing communities is to build a secure web environment to prevent critical information leaks or losses. Many techniques have been developed to secure web applications over the past decade, such as vulnerability scanning, penetration testing, or static source code analysis [1, 2, 3, 5, 6, 9, 11, 12]. Among these technologies, intrusion detection and prevention are two effective methods that can be integrated together to identify malicious activities and to prevent hostile attacks against web systems [1, 9, 11]. These technologies can be used to build an Intrusion Detection and Prevention System (IDPS)—a software system used to detect and prevent existing and new software attacks [19]. This thesis explores how Autonomic Computing technologies and Aspect-Oriented mechanisms can be used to design a new intrusion detection and prevention framework for Java web applications.

This thesis is also part of an industrial collaborative research project entitled *Logging, Monitoring and Diagnosis Systems for Enterprise Software Application* directed by Mankovskii, Müller, Kontogiannis, Mylopoulos, and Wong. CA Canada, Inc. is the industrial partner of this project. “The goal of this research project is to investigate concepts, methods, and prototype tools to assess, evaluate, and evolve the event logging, monitoring, and diagnosis capabilities of selected CA Inc. enterprise applications [40].”

1.2 Approach

IDPSs are used to monitor system activities at run-time, to identify malicious events, and to determine system intrusions or faults. This system can also "respond to a detected threat by attempting to prevent it from succeeding [19]." According to its main functionalities and operational methods, an IDPS could be designed and implemented as a self-adaptive system, which is defined as "a system that continually (at run-time) monitors its success in achieving its intended goal," and modifies itself "in an attempt to do better at its assigned tasks when it is found to be doing poorly" [22]. Feedback loops are the core design elements and the key features of self-adaptive systems. A feedback loop with sensors and effectors is designed to adapt to changing environments in self-adaptive software-intensive systems [23].

To implement such feedback loops, IBM proposed the notion of an autonomic element [24]. An autonomic element can be used to design software systems with adaptive mechanisms [26]. For example, an autonomic element can be employed to monitor and analyze HTTP requests for a web application system where a special response will be sent by this element if a malicious activity occurs. The frequency of the responses, which indicates the number of malicious activities or run-time check violations, could be examined and then used to assess system's safety and to determine the occurrence of system intrusions or failures. Such an autonomic element can be used to implement a system that monitors HTTP transactions.

IDPSs are designed to monitor activities and change states of an existing web system. The general requirements for designing an IDPS are not to modify the system design, model, or code. Aspect-Oriented Programming (AOP) is an excellent candidate

to achieve these requirements [27]. By applying the AOP technology, developers can dynamically modify the static object-oriented model to create a system that needs to fulfill new requirements without modifying the original static model. AOP technology can be applied at compile-time and at run-time to solve the security issues of web applications [28].

Except using AOP technology, we could also apply the Intercepting Filter, which is one of J2EE core patterns, to achieve the requirements for designing an IDPS. The Intercepting Filter pattern wraps existing application resources with a filter that intercepts the reception of a request and the transmission of a response. An intercepting filter can pre-process or redirect application requests, and can post-process or replace the content of application responses [29]. However, a filter has to be specified in the web.xml file inside the <filter> element, along with a corresponding <filter-mapping> that maps a filter to a request pattern. This means that developers have to recreate the web.xml file in an existing web application in order to apply the filter. Unlike Intercepting Filter, AOP technology does not require the developer to modify code and resource files of the existing web applications.

This thesis presents the design of an Aspect-Based Intrusion Detection and Prevention System (AB-IDPS) using AOP and Autonomic Computing technologies to detect and prevent web attacks in web environments. The architecture of this design provides the framework capabilities of self-management. Under this regime, various known web attack methods can be added into the system knowledge base while unknown and potential malicious activities could be detected and recorded for future analysis. The implementation of this thesis focuses on building a prototyping tool with the goal to

detect and prevent two of the most common attacks, Cross Scripting Site (XSS) and SQL Injection. Other parts of this design are left for future work.

1.3 Outline of the Thesis

Chapter 2 provides background knowledge for this thesis, such as web application technologies, web security vulnerabilities, intrusion detection and prevention, AOP, and Autonomic Computing. Chapter 3 describes related work in the area of detecting and preventing web attacks. Chapter 4 discusses the AB-IDPS framework design and its work flow scenarios. Chapter 5 introduces the prototyping tool based on this design framework. Chapter 6 presents the experiment conducted with the prototyping. Chapter 7 concludes the thesis with contributions and future work.

Chapter 2 Background

The purpose of this chapter is to introduce relevant background for the, including web application technologies, web security vulnerabilities, Intrusion Detection and Prevention System, Aspect-Oriented Programming (AOP), and Autonomic Computing (AC).

2.1 Web Application

This research focuses on intrusion detection and prevention in a web application built on Java 2 technologies. A web application is a software application that provides a platform for users to access the web using a web browser via a network such as the Internet or an intranet. A web browser that requests web services or accesses a webpage is called a web client. A web application is also considered a dynamic extension of a web server or an application server. In response to HTTP requests, it generates a static web page with various types of markup languages (e.g. HTML or XML) and dynamic content at run-time [29].

Web components are the key elements of a web server built with Java 2 technologies. A web server can have several web components that provide the dynamic extension capabilities. Web components are Java Servlets, JSP pages, or web service endpoints. Servlets are a set of Java classes that dynamically process requests from web clients and construct contents in response to those requests. JSP pages are text-based documents that process web requests in the same way as Servlets, but dynamically generate web pages based on HTML, XML, or other document types. In the Java 2 platform, the Java Servlet technology is the foundation of all the web application

technologies. Web components are supported by web containers. A web container is a run-time platform that provides services, such as request dispatching, security, concurrency, and life-cycle management, to web components. It also provides web components access to Application Programming Interfaces (APIs) such as naming, transactions, and email [29].

The interaction between a web client and a web application is illustrated in Figure 2-1. The interaction process is as follows:

1) The client sends an HTTP request to the web server.
2) The web server converts the request into an HTTPServletRequest object. This object is delivered to a web component, which can interact with JavaBeans components or a database to generate dynamic content.
3) The web component can then generate an HTTPServletResponse or it can pass the request to another web component.
4) Eventually a web component generates an HTTPServletResponse object.
5) The web server converts this object to an HTTP response and returns it to the client.

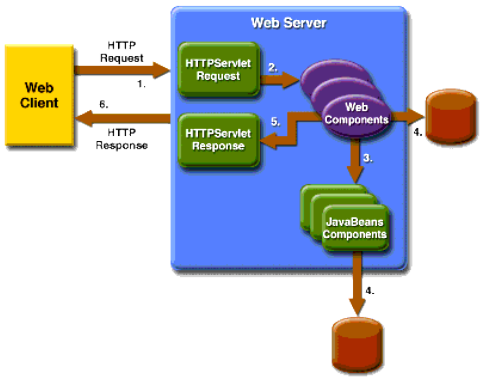


Figure 2-1 Java Web Application Request Handling

A web application is usually divided into logical chunks called “tiers” or “layers”. The most common structure for web applications is the three-tiered structure. The three tiers are called *presentation*, *application* and *storage*, respectively. A web browser is the first tier, the presentation layer. An application engine using Java JSP and Servlet technologies is the middle tier, the application or business layer. A database is the third tier, the storage layer or the persistent layer. The process of a client request is handled in a web application as follows:

1) The web browser sends requests to the application engine.
2) The engine services the requests by making queries to retrieve or update information against the database.
3) The engine generates responses based on the data returned.
4) The responses are sent back to the web client.

2.2 Web Security Vulnerabilities

During the interaction between a web client and a web application, guarantees are often required to maintain the confidentiality of client information. The vulnerabilities of web security provide open doors for web attackers.

2.2.1 Top 10 Vulnerabilities

In 2002, the Open Web Application Security Project (OWASP) Foundation started to keep track of the discoveries of web security vulnerabilities by the security research community. Subsequently, OWASP published the Top 10 most common web application security vulnerabilities aiming to educate people about the consequences of these vulnerabilities. OWASP also provided basic methods for web clients to protect

against these vulnerabilities. To continue to inspire developers, designers, architects and organizations who worry about the security of web applications, the Top 10 edition is re-written about every three years. It provides a list of the most serious web application vulnerabilities and publishes discussions about how to protect against them [17]. Table 2-1 contains the Top 10 web application vulnerabilities for 2007 and Table 2-2 contains the Top 10 for 2010.

A number of new web attack techniques emerge every year, including those that may be positioned in the Top 10. As a result, a web application may be vulnerable to new types of attacks if the new security technologies are not implemented. As we noted before, security issues must be dealt with during all stages of the application development lifecycle, especially during the requirements, design and implementation phases. With respect to the vulnerabilities caused by newly developed attack techniques, however, tremendous work is required to make changes continually on the applications. To avoid the complexity in code modification, a better solution is needed (i.e., to develop a system separated from the original software).

The implementation of the AB-IDPS framework designed in this thesis focuses on two of the most common web attack techniques: Cross Site Scripting (XSS) and SQL Injection, one of the Injection Flaws. These two types of intrusion can be detected and prevented in the implemented model of the AB-IDPS prototype.

Table 2-1 Top 10 Web application vulnerabilities for 2007

A1 – Cross Site Scripting (XSS)	XSS flaws occur whenever an application takes user supplied data and sends it to a web browser without first validating or encoding that content. XSS allows attackers to execute script in the victim's browser which can hijack user sessions, deface web sites, possibly introduce worms, etc.
A2 – Injection Flaws	Injection flaws, particularly SQL injection, are common in web applications. Injection occurs when user-supplied data is sent to an interpreter as part of a command or query. The attacker's hostile data tricks the interpreter into executing unintended commands or changing data.
A3 – Malicious File Execution	Code vulnerable to remote file inclusion (RFI) allows attackers to include hostile code and data, resulting in devastating attacks, such as total server compromise. Malicious file execution attacks affect PHP, XML and any framework which accepts filenames or files from users.
A4 – Insecure Direct Object Reference	A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter. Attackers can manipulate those references to access other objects without authorization.
A5 – Cross Site Request Forgery (CSRF)	A CSRF attack forces a logged-on victim's browser to send a pre-authenticated request to a vulnerable web application, which then forces the victim's browser to perform a hostile action to the benefit of the attacker. CSRF can be as powerful as the web application that it attacks.
A6 – Information Leakage and Improper Error Handling	Applications can unintentionally leak information about their configuration, internal workings, or violate privacy through a variety of application problems. Attackers use this weakness to steal sensitive data or conduct more serious attacks.
A7 – Broken Authentication and Session Management	Account credentials and session tokens are often not properly protected. Attackers compromise passwords, keys, or authentication tokens to assume other users' identities.
A8 – Insecure Cryptographic Storage	Web applications rarely use cryptographic functions properly to protect data and credentials. Attackers use weakly protected data to conduct identity theft and other crimes, such as credit card fraud.
A9 – Insecure Communications	Applications frequently fail to encrypt network traffic when it is necessary to protect sensitive communications.
A10 – Failure to Restrict URL Access	Frequently, an application only protects sensitive functionality by preventing the display of links or URLs to unauthorized users. Attackers can use this weakness to access and perform unauthorized operations by accessing those URLs directly.

Table 2-2 Top 10 Web application vulnerabilities for 2010

A1 – Injection	Injection flaws, such as SQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data.
A2 – Cross Site Scripting (XSS)	XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping. XSS allows attackers to execute script in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
A3 –Broken Authentication and Session Management	Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, session tokens, or exploit implementation flaws to assume other users' identities.
A4 – Insecure Direct Object Reference	A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.
A5 – Cross Site Request Forgery (CSRF)	A CSRF attack forces a logged-on victim's browser to send a pre-authenticated request to a vulnerable web application, which then forces the victim's browser to perform a hostile action to the benefit of the attacker. CSRF can be as powerful as the web application that it attacks.
A6 – Security Misconfiguration	Security depends on having a secure configuration defined for the application, framework, web server, application server, and platform. All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults.
A7 – Failure to Restrict URL Access	Many web applications check URL access rights before rendering protected links and buttons. However, applications need to perform similar access control checks when these pages are accessed, or attackers will be able to forge URLs to access these hidden pages anyway.
A8 – UnvalidatedRedirects and Forwards	Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

A9 – Insecure Cryptographic Storage	Many web applications do not properly protect sensitive data, such as credit cards, SSNs, and authentication credentials, with appropriate encryption or hashing. Attackers may use this weakly protected data to conduct identity theft, credit card fraud, or other crimes.
A10 – Insufficient Transport Layer Protection	Applications frequently fail to encrypt network traffic when it is necessary to protect sensitive communications. When they do, they sometimes support weak algorithms, use expired or invalid certificates, or do not use them correctly.

2.2.2 Cross Site Scripting (XSS)

2.2.1.1 What is Cross Site Scripting

Cross Site Scripting (abbreviated as CSS or XSS) is one of the most common application level attacks that hackers use to invade web applications today [17]. Often people receive an e-mail with a hyperlink to an online banking site. The e-mail induces a person to use the link with promotion techniques such as the chance to win cash. However, if the person clicks the link and logs on to the site, she or he potentially reveals the logon information to a hacker. Users may not be aware that they have unintentionally executed scripts written by an attacker when they followed the links in disguised or unknown sources, such as web pages, e-mail messages, instant messages, newsgroup postings, or various other media. If the malicious scripts are hidden in a link that looks normal, attackers are able to get full access to the retrieved web page and send data contained in the page back to their own server. For instance, a malicious script can read logon fields in an HTML form from a real server, and then send data, such as user name and password information, to the hacker's server. These types of scripts are used widely by web attackers.

2.2.1.2 How Cross Site Scripting Works

As shown in Figure 2-2, the following steps describe how an XSS attack can steal a user's sensitive information.

1. The user receives an e-mail with a link to Bank.com. The link is embedded in the e-mail and contains an attack script.
2. The user clicks the link and connects to Bank.com. At this point, the user unintentionally sends the script as data to Bank.com.
3. The script is returned by Bank.com, but executed on the user's browser.
4. The user's cookie and session information are sent to Evil.org when the script is executed.
5. Using the user's session information, Evil.og is able to impersonate the user to login to Bank.com and the attack has succeeded.

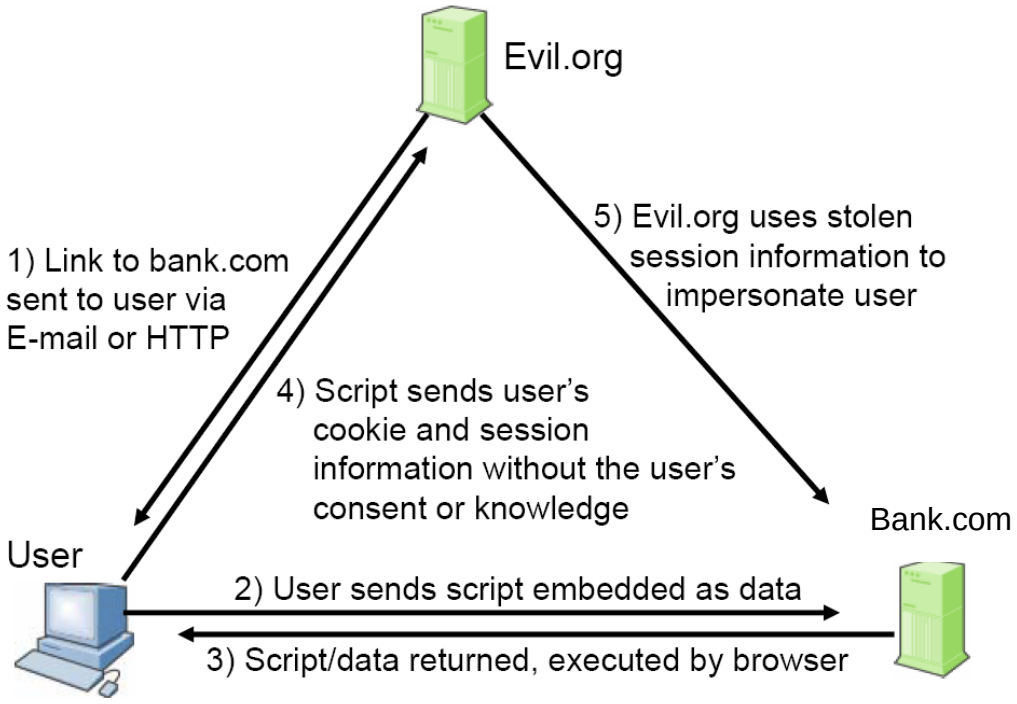


Figure 2-2 Example of a Cross Site Scripting attack

Generally, XSS attacks employ a vulnerable script on a vulnerable site. Such a script reads part of the HTTP request, usually the parameters, but sometimes also HTTP headers or path, and sends it back to the response page. Suppose that this script is named welcome.cgi, and its parameter is “name”. It can be used as follows:

GET /welcome.cgi?name=**Foo%20Bar** HTTP/1.0
Host: bank.com
...

And the response would be:

<HTML>
<Title>Welcome!</Title>
Hi **Foo Bar**

Welcome to Bank.com system
...
</HTML>

Using this script, an attacker is able to create a carefully and maliciously crafted link and lure a user into clicking the link and unintentially executing the script by attaching information to the script. The data attached to the script consist of a Javascript that accesses the cookies for Bank.com. Unfortunately, the Javascript’s security model allows scripts sent from a particular site to access cookies belonging to that site.

Here is how such a link looks like:

http://bank.com/welcome.cgi?name=<script>alert(document.cookie)</script>

The user’s browser, due to clicking the link, generates a request to Bank.com, as follows:

GET /welcome.cgi?name=**<script>alert(document.cookie)</script>** HTTP/1.0
Host: bank.com
...

And the vulnerable site’s response is as follows:

<HTML>
<Title>Welcome!</Title>

Hi **<script>alert(document.cookie)</script>**

Welcome to Bank.com system
...
</HTML>

The user's browser then interprets this response as an HTML page with a piece of Javascript code. This code is executed to access all cookies belonging to Bank.com, and therefore a window will pop-up and show all client cookies belonging to Bank.com.

A real attack program will send these cookies to the attacker, who may need a web site, such as Evil.org, and use a script to receive the cookies. The scripts used by the attacker must be able to access a URL at Evil.org and invoke the cookie reception script with the stolen cookies as a parameter. This way, the attacker can get the cookies from the Bank.com server.

For example, the malicious link may look like this:

http://www.vulnerable.site/welcome.cgi?name=<script>window.open("http://www.attacker.site/collect.cgi?cookie="%2Bdocument.cookie)</script>

And the response page would look like this:

<HTML>
<Title>Welcome!</Title>
Hi
<script>window.open("http://www.attacker.site/collect.cgi?cookie="+document.cookie)</script>

Welcome to Bank.com system
...
</HTML>

The user's browser would load this page, execute the embedded Javascript, and send a request to the attacking script, collect.cgi in Evil.org, together with the value of the cookies of Bank.com uploaded by the user's browser. After retrieving the personal information of the user from the cookies, the attacker is able to impersonate an actual user, and the privacy of the user is breached.

It should be noted that a Javascript pop-up window is usually sufficient to detect whether a site is vulnerable to an XSS attack. If the Javascript's "alert" function is called, there is usually no reason for the "window.open" or "document.cookie" call not to succeed. That is why most examples for XSS attacks use the alert function, which makes it very easy to detect its success.

Cross Site Scripting is one of the most common and dangerous application level attacks. It is an attack that can lead to a total breach of security where sensitive data of users of a particular web site details could be stolen or manipulated. AB-IDPS, the prototyping framework of this thesis, aims to detect and prevent this kind of attack in order to secure the vulnerability of web applications.

2.2.2 SQL injection

"Injection occurs when user-supplied data is sent to an interpreter as part of a command or query. Attackers trick the interpreter into executing unintended commands via supplying specially crafted data. Injection flaws allow attackers to create, read, update, or delete arbitrary data available to the application. In a worst case scenario, these flaws allow an attacker to completely compromise the application and the underlying systems, even bypassing deeply nested firewalled environments [17]."

SQL injection is the most popular type of injection flaw. This technique exploits security vulnerabilities of the database layer of a web application. The malicious SQL commands are usually embedded into parameters that a web application sends to a database. An SQL injection attack occurs when string escape characters embedded in SQL statements are not filtered correctly or a type is handled incorrectly. As a result,

sensitive data stored in a database can be stolen, and the database contents may even be corrupted or destroyed.

2.2.2.1 Incorrectly filtered escape characters

This form of SQL injection occurs when escape characters of user input are not filtered and then carried by an SQL statement into a database. As a result, a malicious manipulation of the statements is performed on the database. Two techniques, tautology and additional declaration and comments, can be used to achieve this kind of SQL injection.

- **Tautology**

The general intent of a tautology-based attack is to inject malicious code in one or more conditional statements so that they always evaluate to be true. Such a tautology is a disjunction in the WHERE clause of a select or update statement. The following line of code demonstrates this vulnerability:

sqlStatement = "SELECT * FROM users WHERE name = '" + userName + "';"

This SQL code will get the records of the specified username from the table called *users*. Unfortunately, if the "userName" variable is crafted in a specific way by a malicious user, the SQL statement may do more than what is expected. For example, the "userName" variable could be ***x' or 'y'='y***, resulting in the following query statement:

SELECT * FROM users WHERE name = 'x' or 'y'='y';

As we can see, the WHERE clause is always true because the evaluation of 'y'='y' is always true. Thus, this select statement returns all rows in the table *users*. Instead of using 'y'='y' or any other characters, we could also use 1=1, which is always true for integer.

The consequence is even worse if the database API allows multiple statements in a query, such as the one below.

x';DROP TABLE users; SELECT * FROM userinfo WHERE 'y' = 'y

The final SQL statement would look like this:

SELECT * FROM users WHERE name = 'x';DROP TABLE users; SELECT * FROM userinfo WHERE 'y' = 'y';

The query results in the selection of all data from the table *userinfo* and also causes the deletion of the "users" table. Essentially it reveals the information of every user in the table.

- **Additional declaration and comments**

The characters of comments can also be used for attacking this vulnerability. In most database systems, the "--" or "#" characters are defined as a comment indicator. An SQL query can be cut into pieces by the comments and its meaning may be changed as well. For instance, the SQL statement below:

SELECT * FORM users WHERE name = 'foo' and password = 'bar'

can be transformed as follows:

SELECT * FORM users WHERE name = 'admin' -- and password = ''

The result of this query is that all the information about the user *admin* in the table *users* even though there is no correct password in the query, because the characters after "--" are interpreted as comments by the database system.

2.2.2.2 Incorrect type handling

This type of an SQL injection occurs when a user input field is not strongly typed or not checked for type constraints. If there is no code to check and validate that the user supplied input is numeric, the attack could take place when a numeric field is to be used

in an SQL statement. For example, in the following statement:

```
sqlString = "SELECT * FROM userinfo WHERE id = " + id_num + ";"
```

id_num is obviously expected to be a number correlating to the field *id*. However, a user may manipulate the statement as they choose if the field is actually defined as a string. For example, setting *id_num* to

```
1;DROP TABLE users
```

And the query would be rendered as follows:

```
SELECT * FROM userinfo WHERE id=1;DROP TABLE users;
```

This statement will drop or delete the table *users* from the database.

SQL injection is one of the most famous attack techniques that aim to attack the database layer of a web application. Several other types of SQL injection, such as Union Query, Blind Injection, and Timing Attacks [17], are not discussed in this thesis. AB-IDPS has been implemented to support the detection and prevention of selected SQL injection types such as the ones illustrated in this section.

2.3 Intrusion Detection and Prevention

In the computing world, intrusion is an incidence of unauthorized access to resources and data of computer systems or networks. “Intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of possible incidents, which are violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices. Intrusion prevention is the process of performing intrusion detection and attempting to stop detected possible incidents [19].” An intrusion detection system (IDS) is a software system that is able to execute the intrusion detection process. An intrusion prevention

system (IPS) is an extension of an intrusion detection system. It is a software system that has all the capabilities of an intrusion detection system and can also perform attempts to stop intrusion actions. An intrusion detection system (IDS) is usually a passive system while an intrusion prevention system (IPS) is considered a reactive system.

From a functionality perspective, an IPS can also be called intrusion detection and prevention systems (IDPS). It primarily focuses on identifying possible intrusions, recording information about them, attempting to stop them, and reporting them to system administrators. In addition, organizations use IDPSs for some other purposes, such as identifying incidences with security policies, logging existing attacks and threats, and deterring individuals who are violating security policies. For nearly every organization in the world, IDPSs are increasingly necessary for security infrastructure [19].

A typical IDPS should have four fundamental components: *sensor or agent*, *management server*, *database server*, and *console*. Sensors and agents monitor and analyze both inside and outside activities. A management server receives information from the sensors or agents and manages them. Also, a management server can analyze the event information from the sensors or agents and can identify events, which an individual sensor or agent cannot. A database server is a central repository for event information from sensors, agents, and management servers. A console is a user interface that provides communications among the IDPS's users, administrators, and backend sub-systems [19].

It is critically important for a web application system to design the security mechanisms to prevent unauthorized access to its resources and data. However, it is not realistic to rely on developers to take into account all security issues during the development cycle. Therefore, an IDPS should be used separately (i.e., outside of the web

application) to monitor network traffic, particularly on the HTTP layer, and take necessary actions for security breaches. Our AB-IDPS framework is designed and implemented to achieve this goal.

2.4 Aspect-Oriented Programming (AOP)

2.4.1 Introduction

In computing, Object-Oriented Programming (OOP) is a programming paradigm that uses the underlying object model to provide a better solution to real domain problems [27]. In 1996, Gregor Kiczales and his team recognized that OOP techniques are insufficient to capture all the important design decisions in many programming problems clearly. These programming problems cannot be solved easily with a procedural approach. Gregor Kiczales and his team analyzed the reason of "why some design decisions have been so difficult to cleanly capture in actual code.[27]" They introduced AOP to "clearly express programs involving such aspects, including appropriate isolation, composition and reuse of the aspect code."

The motivation for aspect-oriented programming approaches is to solve the problems, like code scattering and tangling, that are not well captured by traditional programming methodologies. Code scattering means that the problem code is spread out over multiple modules. For example, developers may have to scan and modify several source files when they want to fix a bug. This process is error-prone and difficult to deal with in traditional programming languages. The code scattered around several classes might also be redundant. Code tangling means that the problem code is intermixed with other code. The tangled code cannot be encapsulated by separate modules using regular

techniques [27]. The concerns of scattered and tangled code in a program are called *crosscutting concerns*. "Aspect-oriented programming is a way of modularizing crosscutting concerns much like object-oriented programming is a way of modularizing common concerns [13]." Logging code of a software system is a common example of a crosscutting concern. Logging code *crosscuts* all logged classes and methods, and thus affects each logged part of a program.

Another common example of programming concerns is security, and AOP is able to address it in a modular way. Through AOP languages and tools, security issues in a software system can be compartmentalized at compile-time or at run-time without changing the business part of the software. Among those languages and tools, AspectJ is the most widely known and commonly used one.

2.4.2 AspectJ

AspectJ is an implementation of AOP programming paradigm for Java. It is a seamless aspect-oriented extension to Java with clean modularization of crosscutting concerns [13]. AspectJ provides a new concept, *join point*, and several new constructs: *pointcuts*, *advice*, *inter-type declarations* and *aspects*. In AspectJ, the program flow is affected dynamically by *pointcuts* and *advice* and a program's class hierarchy can be modified statically through *inter-type declarations*. *Aspects* encapsulate these new constructs in the code. They are defined as follows [13]:

- A *join point* is a well-defined point in the program flow.
- A *pointcut* picks out certain join points and values at those points.

```
pointcut set() : execution(* set*(..)) ;
```

This *pointcut* matches a method-execution join point, if the method name starts with "set" and there is exactly one argument of any type.

- A piece of *advice* is a piece of code that is executed when a join point is reached.

```
after () : set() {
        Display.update();
}
```

The *advice* example means: "if the set() pointcut matches the join point, run the code Display.update() after the join point completes."

- AspectJ also has different kinds of *inter-type declarations* that allow the programmer to modify a program's static structure, namely, the members of its classes and the relationship between classes.

```
Aspect VisitAspect {
        Void Point.acceptVisitor(Visitor v) {
                v.visit(this);
        }
}
```

This example code adds the acceptVisitor method to the Point class.

- *Aspects* are the unit of modularity for crosscutting concerns. They behave somewhat like Java classes, but may also include *pointcuts*, *advice* and *inter-type declarations*.

AspectJ performs two logical steps: “It first combines the individual concerns using the weaving rules, and then converts the resulting information into executable code. The process that combines the individual concerns according to the weaving rules is called *weaving* and the processor doing this job is called a *weaver* [33].”

In the prototype of this thesis, AspectJ is one of the key technologies used to achieve the final design goal, and it plays an important role in the prototype implementation.

2.5 Autonomic Computing (AC)

2.5.1 Introduction

The complexity of computing systems has been rapidly and dramatically growing over the past decades. The problem of complexity has affected system developers, administrators and end users in many aspects, such as system management and maintenance. Contributing to this complexity crisis, IBM proposed an approach for the development of systems and applications to self-managed systems by publishing an architectural blueprint [24]. Autonomic computing is a systematic approach to the coordination and automatic management of computing systems. The autonomic computing initiative represents a holistic, goal-oriented approach to designing computer systems [31]. Its primary goal is to develop computer systems capable of self-management.

IBM advocated four fundamental capabilities for self-management [24]:

- Self-configuration: the ability to configure and re-configure itself to react to varying and unpredictable changes in its operational environments;
- Self-optimization: the ability to detect and optimize its resource and operations;
- Self-protection: the ability to prevent various attacks from both inside and outside of the system;

- Self-healing: the ability to detect problems and failures and to recover from them.

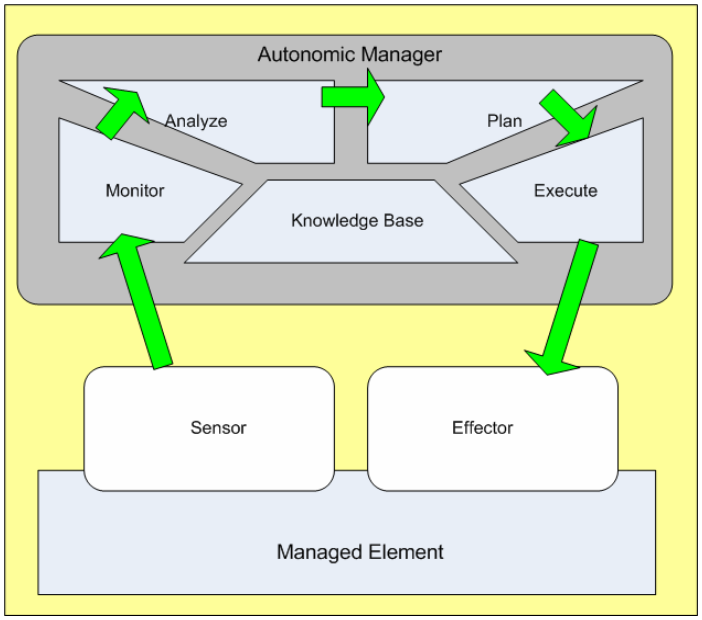


Figure 2-3 Autonomic Element [32]

2.5.2 Autonomic Computing Elements

In the IBM architectural blueprint, an autonomic element is defined as a fundamental building block for designing self-configuring, self-healing, self-protecting and self-optimizing systems [24]. As depicted in Figure 2-3, an autonomic element consists of an autonomic manager, a managed element, sensors, and effectors. The autonomic manager uses sensors and effectors to interact with the managed element. The autonomic element is composed of a monitor, an analyzer, a planner, an executor and a shared knowledge base. The monitor senses the managed process and its own behaviour in order to detect service delivery failures, and stores relevant events in the knowledge base for future analysis. The analyzer compares event data against patterns in the

knowledge base to diagnose the symptoms, and stores the symptoms for future reference in the knowledge base. The planner interprets the symptoms and plans proper fault remediation strategies. These plans are executed by the executor in order to restore the normal system behaviour through the effectors. “To facilitate collaboration among autonomic elements, the control and data of manageability interfaces are standardized across managed elements and autonomic building blocks [23].”

In our prototype, we have implemented an autonomic element that is able to monitor HTTP transactions between clients and the web application, analyze HTTP requests and responses, and execute the planned strategies to prevent malicious attacks.

2.6 Summary

This chapter reviewed useful selected background information for this thesis, such as the components of a web application, the Top 10 web vulnerabilities as of 2007 and 2010, the basics of intrusion detection and prevention, the mechanism of AOP and AspectJ, and architectural components of self-managing.

Chapter 3 Related Work

In this chapter we review and discuss a number of proposed solutions to security problems associated with web applications in recent years. These solutions can be categorized into two classes: The first one focuses on detecting and blocking web-based attacks and the other one on identifying vulnerabilities in the implementation of a web application prior to the deployment of the application. These classes employ different approaches to achieve the web security solutions.

3.1 Attack Detecting and Blocking

There are two types of solutions in the first class: One analyzes the requests sent to web applications [2, 3, 5, 7, 9, 10, 11]; another analyzes the content delivered by the applications to the users [4, 8]. These solutions do not require any modifications to the protected application. However, they significantly impact the system's performance and may result in wrong detections and block legitimate traffic.

Kruegel and Vigna have proposed an intrusion detection system that detects attacks against web servers and web-based applications using a number of different anomaly detection techniques [2]. A significant characteristic of the system is the correlation of the server side programs referenced by client queries and the parameters contained in these queries. The application-specific characteristics enable the system to perform more effectively with a reduced number of false positives. At the same time, the automatic derivation of the association between the parameter and web applications enables the system to be deployed in very different application environments without time-consuming tuning and configuration.

Scott and Sharp focused on the problem of application-level web security in their paper [3]. The paper presents a scalable structuring mechanism that facilitates the abstraction of security policies from large web applications developed in heterogeneous multi-platform environments. A tool has also been designed to assist programmers in developing flexible secure applications that can be used to handle a wide range of common attacks and to report results of the implementation and experiences arising from the implementation.

Kirda et al. present Noxes, probably the first client-side solution that acts as a web proxy to reduce cross-site scripting attacks [4]. Noxes can be applied to protect against information leakage from the user's environment effectively. It uses rules generated both automatically and manually, therefore requires minimal user interaction and customization effort.

Hermosillo et al. presented AProSec, implemented with the AspectJ language and the JBoss AOP framework, for detecting SQL injection and Cross Scripting Site (XSS) [5]. Their experimentations show the advantage of changing security policies at run-time on run-time platforms such as JBoss AOP.

Almgren et al. presented an intrusion-detection tool that focuses on web server attacks, and explained why such a tool is necessary [7]. An interesting aspect of the tool is its ability to keep track of suspicious hosts, which can assist in identifying new attacks. The tool is flexible in that it allows detection of a wide variety of malicious behaviour considering the history of different types of attacks on a host basis. Furthermore, the tool includes mechanisms that can be applied to reduce the rate of false alarms.

Jovanovic et al. presented a solution that provides protection from Cross Site Request Forgery (XSRF) attacks [8]. Based on a server-side proxy, the detection and prevention of XSRF attacks are transparent to users as well as to the web application itself. Their paper demonstrates that the prototype can be used to secure a number of popular open-source web applications, without negatively affecting the performance of the applications.

Uddin et al. provided an intrusion detection and prevention architecture for Component-Based Software (CBS) based on an aspect-connector, ACIR (Aspect Connector for Intrusion Response) [9]. In this approach, aspects contain pointcuts and two types of advices applicable to the pointcuts: signature advice is used to detect intrusions, and action advice is used to prevent intrusions. A prototype of this architecture is evaluated against intrusions included in the Web Application Security Consortium (WASC) intrusion list. The significant characteristics of this approach are the encapsulation, reusability, and modularity of the architecture due to the use component interfaces as join points.

Haldar et al. proposed a dynamic solution that tags and tracks user input at run-time and prevents malicious execution of the program [10]. The implementation of the solution can be applied to Java class files without source code. Benchmark evaluations show that this technique can prevent a number of attacks with negligible overhead.

Vigna et al. presented an intrusion detection system, called WebSTAT that analyzes web requests to look for evidence of malicious behaviour [11]. This novel system provides a sophisticated language to describe multi-step attacks in terms of web states and transitions. In addition, the system supports the integrated analysis of network

traffic sent to the server host, the operating system-level audit data produced by the server host, and the access logs produced by the web server. This system can achieve more effective detection of web-based attacks by correlating different streams of events.

3.2 Vulnerability Identification before Deployment

The approaches in the second class of solutions for web security utilize static and dynamic analysis techniques to identify vulnerabilities in web application [1, 6, 12]. Most of the approaches in this class are based on the assumption that the source of vulnerabilities in web applications is the insecure data flow accessing the web applications. Consequently, these techniques are designed in an attempt to identify when the input data is used in security critical operations without being checked and sanitized before accessing the application.

Balzarotti et al. developed a novel vulnerability analysis approach that is able to identify sophisticated multi-step attacks against the application's workflow that were not addressed by other vulnerability analysis approaches [1]. The attack identification is achieved by characterizing both the extended state and the intended workflow of a web application. By doing this, the approach takes into account both inter-module relationships and the interaction of an application's modules with back-end databases. The prototype tool, called MiMoSA, was evaluated and shown to have the ability to identify known and unknown vulnerabilities on several applications,

Jovanovic et al. addressed the problem of vulnerable web applications and proposed a solution using flow-sensitive, interprocedural and context-sensitive data flow analysis to discover vulnerable points in a program [6]. In their solution, alias and literal

analysis are also employed to improve the correctness and precision of the vulnerability identification results. The approach aims at the general class of security problems that are vulnerable to common attacks such as SQL injection, cross-site scripting, or command injection. The detection of cross-site scripting vulnerabilities in PHP scripts is illustrated in the evaluation of Pixy, the open source prototype implementation of this approach, The result shows that 15 previously unknown vulnerabilities have been reported in three web applications, and 36 known vulnerabilities have been detected in three other web applications.

Halfond and Orso presented and evaluated a new SQL injection detection and prevention technique that uses a model-based approach to detect illegal queries before they are executed on the database [12]. In the static part of the technique, program analysis is used to automatically build a model of the legitimate queries that could be generated by a web application. In the dynamic part of the technique, run-time monitoring is used to inspect the dynamically-generated queries and to check them against the statically-built model. A tool called AMNESIA was developed to evaluate the technique on seven web applications. The results of the evaluation show that, among a large number of both legitimate and malicious inputs, AMNESIA was able to stop all of the attempted attacks without generating any false positives.

3.3 Summary

In recent years, many solutions were proposed and evaluated in response to web security problems. We categorized these solutions into two classes: attack detecting and blocking and vulnerability identification. The approaches of the first class focus on attack analysis, and employed techniques and mechanisms to improve the effectiveness of

intrusion detection and prevention. Vulnerability identification focuses on the security problems that cause a web application to be vulnerable to particular attacks. The solutions for this problem apply methods, such as static and dynamic analysis, to analyse input data flow originating from the outside of a web application. Both correctness and precision of the identification are evaluated in those proposed solutions.

Chapter 4 AB-IDPS Framework

As discussed in the background chapter, a typical Intrusion Detection and Prevention System (IDPS) consists of four fundamental components: sensor or agent, management server, database server, and console [19]. Our Aspect-Based Intrusion Detection and Prevention System (AB-IDPS) is designed as a type of IDPS employing AOP and AC technologies. We designed our AB-IDPS to be a framework with all components that an IDPS should have: a sensor or agent, a management server, a database server, and a console. We applied one aspect for sensing and effecting HTTP requests and responses; the framework will not work without it. This is the reason for using the term aspect-based in the title of the thesis. The following table shows the correspondence between the components of a typical IDPS and our AB-IDPS.

Table 4-1 AB-IDPS Components VS Typical IDPS Components

Components of a typical IDPS	Components of AB-IDPS Framework
Sensor or agent	An aspect of HTTP Servlets
Management server	Autonomic elements (monitor, analyzer, planner, and executor)
Database server	Knowledge Base
Console	A user interface (web-based)

4.1 Architecture

The goal of the AB-IDPS framework is to help IT professionals build web attack detection and prevention systems for Java web applications and web services. We introduce the description of our architecture with a diagram showing the main

components of system as depicted in Figure 4-1 below. The diagram also illustrates the role that components of the Architecture Framework play and the connections among those components. The architecture of the system is described in terms of the components and their interactions. A software component is a part of the system which performs a single function and has a well defined interface.

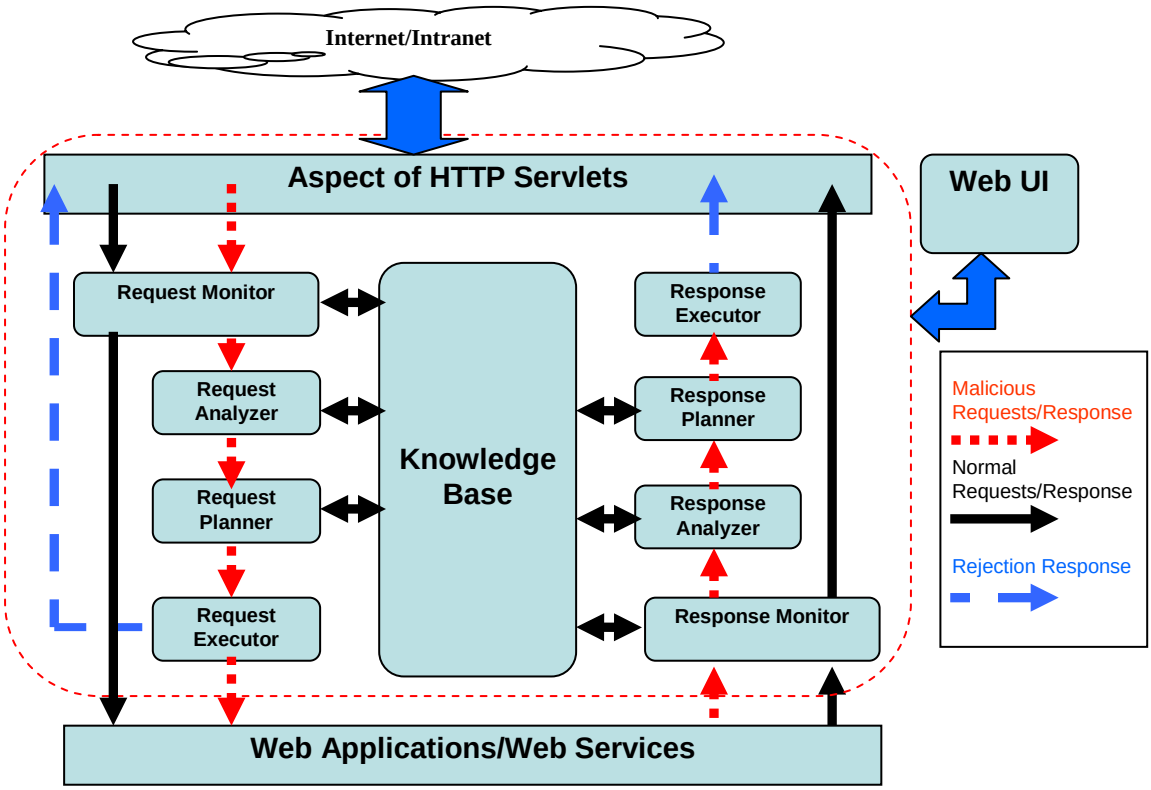


Figure 4-1 AB-IDPS System Architecture

4.2 Aspect of HTTP Servlets

This component is the first component that incoming HTTP traffic will encounter. From the IDPS perspective, it is considered a sensor or an agent. It is designed to collect HTTP request and response data, but filter out others in the HTTP layer. The collected data is then handed over to autonomic elements.

4.3 Autonomic Elements

A general approach to detect web attacks is to monitor HTTP requests. Our approach is to monitor both HTTP requests and responses. The HTTP response contains the data that are sent back to the web client from the web application. Through analyzing the HTTP response, the AB-IDPS framework can learn whether the web application has abilities to protect itself from certain web attacks. Therefore, the framework needs two feedback loops, one for HTTP requests and another for HTTP responses. For this reason, we designed two separate autonomic elements for HTTP requests and responses.

Based on the defined rules, the feedback loop for an HTTP request is able to reject the requests with any kinds of known web attack. Similarly, the feedback loop for an HTTP response is able to revise the response content if the response contains any malicious content. However, to achieve the long term goals, the feedback loops need to assume more responsibilities for learning and analyzing the activities between web applications and clients to be able to detect unknown types of web attack techniques. For example, the monitor logs every communication activity within 24 hours, and the diagnostic data are stored in the knowledge base. The analyzer determines whether there are malicious behaviours based on analyzing these data. The planner can devise a plan to either notice the executor to block similar behaviours from the client or plan a longer term observations to make more informed decisions later on.

4.3.1 Request/Response Monitor

- Request Monitor (RTM)

  Request Monitor (RTM) monitors the HTTP requests which end users of the applications or services send over the Internet. There are two categories of

HTTP requests: normal and abnormal requests. They are defined in the component named Knowledge Base. Based on the information in Knowledge Base, RTM takes different actions to handle these requests. The abnormal requests are passed to the Request Analyzer for further analysis, while the normal requests are transferred to Web Applications or Services directly.

- Response Monitor (RSM)

  Response Monitor (RSM) monitors the HTTP responses which the applications or services send back to the users. Similar to HTTP requests, responses can be classified as normal or abnormal responses. They are also well defined in the Knowledge Base, and RSM handles these responses differently. The abnormal responses are passed to the Response Analyzer for further analysis, while the normal responses are sent back to the users without filtering.

4.3.2 Request/Response Analyzer

- Request Analyzer (RTA)

  Request Analyzer (RTA) provides deep analysis of HTTP abnormal requests based on the information in the Knowledge Base. The analysis results, which will be used by the Request Planner (RTP) or Response Analyzer (RSA) later, are stored in the Knowledge Base.

- Response Analyzer (RSA)

  Similarly, Response Analyzer (RSA) provides deep analysis of HTTP abnormal responses based on the RTA analysis results in the Knowledge Base. RSA will check whether the responses include some sensitive

information, such as administrator information. The analysis results are stored in the Knowledge Base for the future uses.

4.3.3 Request/Response Planner

Request Planner (RTP) and Response Planner (RSP) determine an appropriate procedure of action according to the analysis results in the Knowledge Base. RTP decides what action should be taken next and passes the decision to the Response Executor. Similarly, RSP decides what action should be taken next and passes the decision to the Response Executor.

4.3.4 Request/Response Executor

Request Executor (RTE) and Response Executor (RSE) take appropriate actions based on the decisions made by RTP and RSP. For example, the rejection HTTP responses will be sent back to users if RTP knows that the application or service does not filter the malicious scripts.

4.3.5 Knowledge Base

The Knowledge Base provides well defined rules, statistics and dynamic analysis information for monitors, analyzers, and planners. For example, the monitors use the rules to make judgments whether the requests or response are normal or not. A monitor can also collect all event data within a time window and store them in the Knowledge Base. Storing event data in Knowledge Base provides enough valuable information for analyzers. All the decisions made by Planners are also based on the information stored in the Knowledge Base. Thus, the Knowledge Base is the key component in this framework design.

4.4 Web-Based User Interface

This component is the visualization part of this framework and utilizes a web application or web services. It could be considered the console in an IDPS. It could provide administrative functions, AB-IDPS health monitor and control, Knowledge Base management, and new type intrusions input by administrators.

4.5 AB-IDPS Work Flows

4.5.1 Handling Normal Requests

- ***Scenario 1: Normal Requests and Responses***

Request Monitor (RTM) considers the received request normal, and the server sends out a normal response. The work flow is outlined in the following steps (Figure 4-2):

1) RTM receives the request.
2) RTM checks with Knowledge Base and considers that the request is normal.
3) RTM passes the request to the server.
4) The server sends out the response.
5) RSM checks with Knowledge Base and considers that the response is normal.
6) RSM sends the response to users.

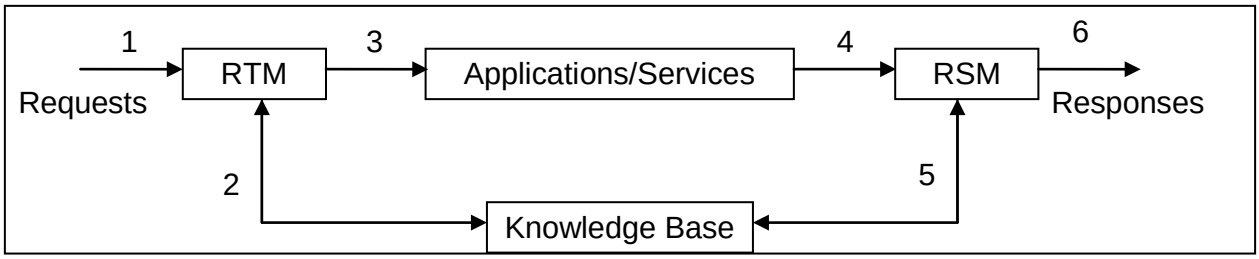


Figure 4-2 Handling Normal Requests and Responses

❖ ***Scenario 2: Normal Requests and Abnormal Responses***

Request Monitor (RTM) considers the received request normal, but the server send out the response considered abnormal. The work flow is outlined in the following steps (Figure 4-3):

1) RTM receives the request.
2) RTM checks with Knowledge Base and considers that the request is normal.
3) RTM passes the request to the server.
4) The server sends out the response.
5) RSM checks with Knowledge Base and considers that the response is abnormal.
6) RSM passes the responses and the information to RSA.
7) RSA record the information to the Knowledge Base.
8) RSP tells RSE to send out rejection responses.
9) RSE sends out rejection responses.

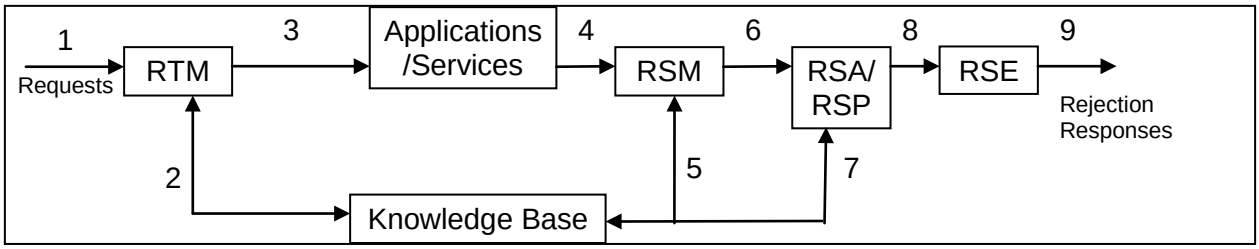


Figure 4-3 Handling Normal Requests and Abnormal Responses

4.5.2 Handling Abnormal Requests

The actions that AB-IDPS takes to handle abnormal requests are different due to the applications or services' behaviours and reactions when receiving malicious requests. AB-IDPS will learn the behaviours and reactions based on the server's responses and take distinct actions.

4.5.2.1 Handled by Servers

The work flow handling abnormal requests for the first time is not same as the second or subsequent times, even though the server has dealt with the malicious requests.

- ***Scenario 3: Receive abnormal requests for the first time***

  1) RTM receives the requests.
  2) RTM checks whether the requests is normal.
  3) RTM passes the abnormal requests to RTA.
  4) RTA checks whether the requests is received at the first time. If so, it will do deep analysis and record the results in the Knowledge Base. RTA will pass the request to RTP/RTE if the request causes a state change that corrupts the web application (e.g., dropping tables) and the request will be rejected immediately.

5) RTP passes the requests to the server.
6) The server sends out the normal responses.
7) RSM checks whether the responses is normal.
8) RSM sends the normal responses to users and passes the information to RSA.
9) RSA record the information to the Knowledge Base.

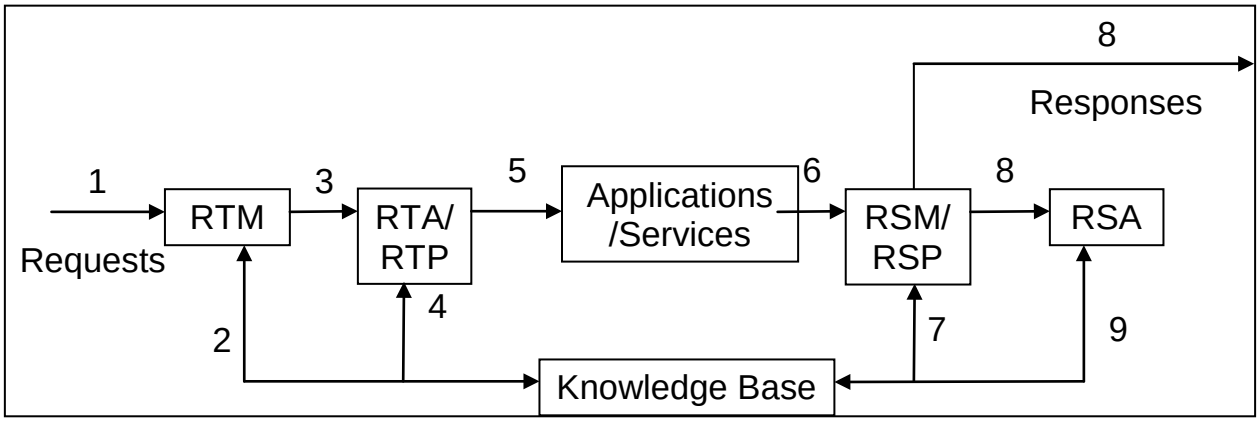


Figure 4-4 Abnormal Requests Handled by Servers (First Time)

❖ ***Scenario 4: Receive abnormal requests at subsequent times***

This is same work flow as for handling normal requests (Figure 4-2).

4.5.2.2 Not Handled by Servers

❖ ***Scenario 5: Receive abnormal requests for the first time***

1) RTM receives the requests.
2) RTM checks whether the requests is normal.
3) RTM passes the abnormal requests to RTA.

4) RTA checks whether the requests is received at the first time. If so, it will do deep analysis and record the results in the Knowledge Base. RTA will pass the request to RTP/RTE if the request causes a state change that corrupts the web application (e.g., dropping tables) and the request will be rejected immediately.
5) RTP passes the requests to the server.
6) The server sends out the abnormal responses.
7) RSM checks whether the responses is normal.
8) RSM passes the responses and the information to RSA.
9) RSA record the information to the Knowledge Base.
10) RSP tells RSE to send out rejection responses.
11) RSE sends out rejection responses.

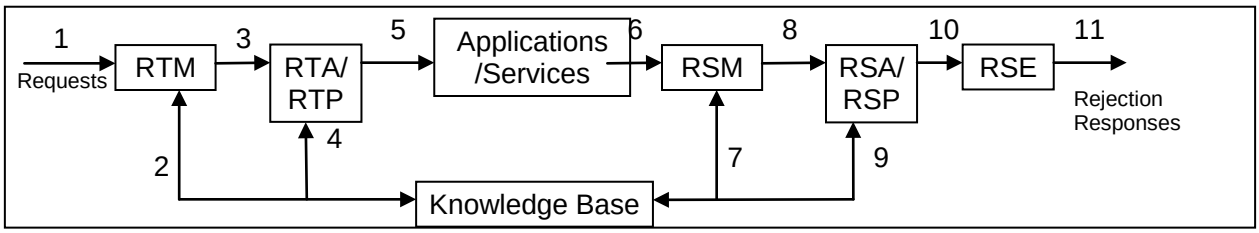


Figure 4-5 Abnormal Requests Not Handled by Servers (First Time)

❖ ***Scenario 6: Receive abnormal requests for subsequent times.***

1) RTM receives the requests.
2) RTM checks whether the requests is normal.
3) RTM passes the abnormal requests to RTA.

4) RTA checks whether the abnormal requests is received at the first time. If it is not the first time, RTA records the results in the Knowledge Base.
5) RTP passes the rejection decision to RTE.
6) RTE sends out the rejection responses.

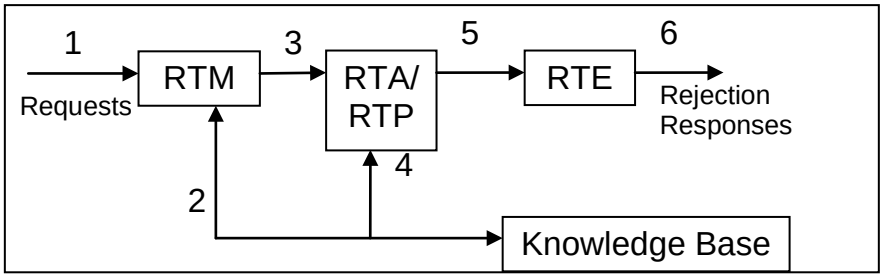


Figure 4-6 Abnormal Requests Not Handled by Servers (Rest Time)

4.6 Summary

AB-IDPS is designed as a type of IDPS. This framework employs AOP and AC technologies to improve maintainability, flexibility and scalability of an IDPS. The aspect component acts as an agent that collects HTTP request and response data, and then hands the request over to autonomic elements. The autonomic component analyzes the HTTP response and enables the ability of the AB-IDPS framework to learn the security capacity of a web application. The key element of the framework, the Knowledge Base, provides well defined rules, statistics and dynamic analysis information for monitors, analyzers, and planners. We also presented the system work flows in different scenarios. The implementation of this design is described in detail in the next chapter.

Chapter 5 Implementation of AB-IDPS

The previous chapter introduced the high level design of the AB-IDPS framework. We implemented a prototype tool based on this design to detect and prevent intrusions, such as Cross Site Scripting and SQL Injection, for Java-based web applications. The prototype is implemented in Java and consists of three core modules: AOP Module, Autonomic Module, and Knowledge Base Module—leveraging several existing technologies and libraries.

5.1 Logic Modules

5.1.1 AOP Module

This module implements the technique to instrument Java web applications using AOP. It gets an HTTP request or response from a Servlet and instruments it with a call to the autonomic monitor. We implemented this module using AspectJ, a seamless aspect-oriented extension to the Java programming language that enables clean modularization of these 'crosscutting concerns' [13].

5.1.2 Autonomic Module

This module implements four components of an autonomic element. It includes Monitor, Analyze, Plan and Execute. Each component has the basic functions described under autonomic computing architecture of the background section. For the implementation of this module, we developed abstract and concrete classes for each autonomic element for both HTTP request and response. The autonomic module is able

to analyze Java Servlets as well as JSP pages. The Knowledge Base of an autonomic element is implemented separately in the other module.

5.1.3 Knowledge Base Module

This module implements the Knowledge Base of an autonomic element. It includes two different functional parts: dynamic sub-module and static sub-module.

5.1.3.1 Dynamic Sub-module

This sub-module implements a business decision maker using an external API. At run-time, it takes a query string as input, and checks the query against the knowledge data. The knowledge data can also be changed dynamically at run-time. For this module, we leveraged two business logic integration libraries. One is JBoss Drool Expert, a unified and integrated platform for Rules [14]. Another is Esper, a component for Complex Event Processing (CEP) and Event stream Processing ESP written in Java [15]. We can select any one as a knowledge base implementation through a property file.

5.1.3.2 Static Sub-module

This module implements a static Knowledge Base for an autonomic element. It loads the predefined static knowledge data from a database when the system starts for the first time. The static data can be input manually, or produced by the dynamic sub-module when detecting intrusion. With this module, all static data are stored in a database. For this module, we leveraged Hibernate, “a powerful, high performance object/relational persistence and query service” [16]. Also, a Java caching system called Apache JCS is used to improve the performance of reading data [34].

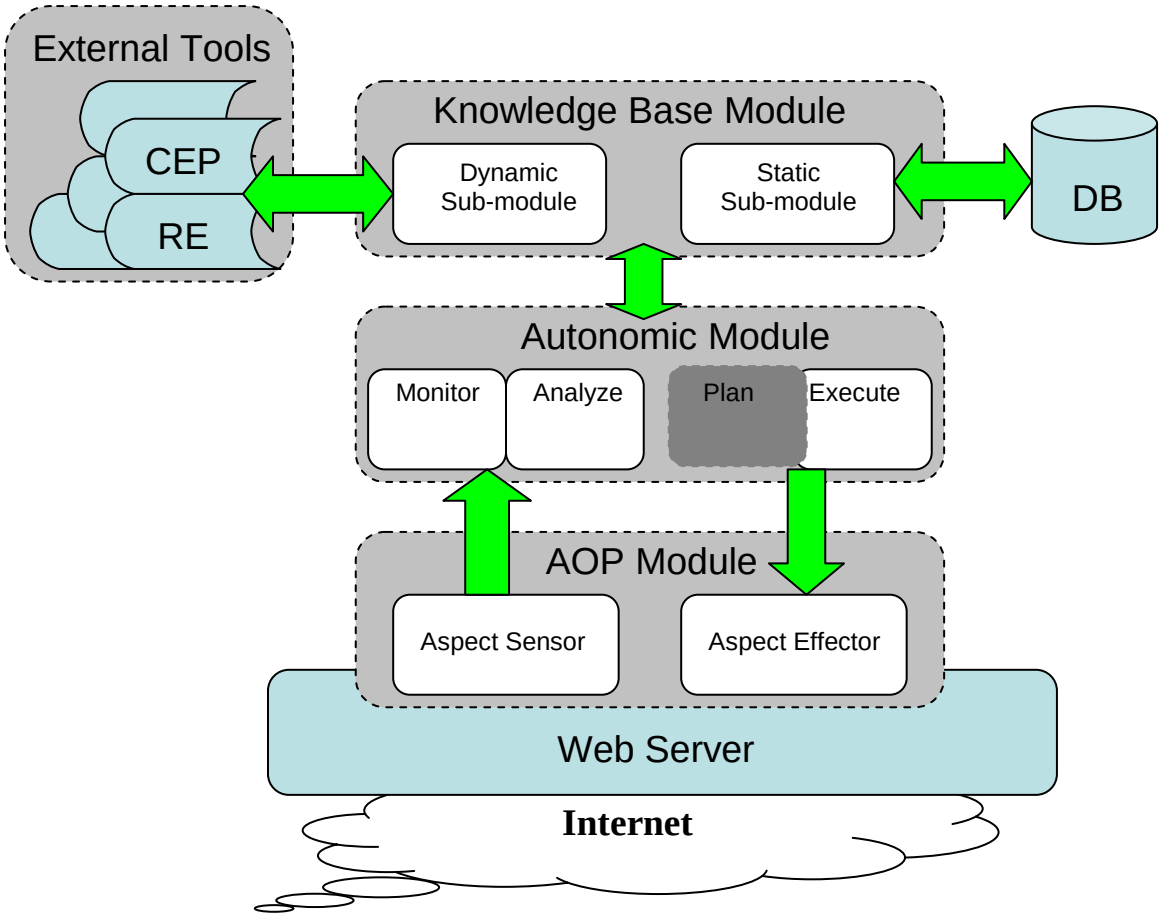


Figure 5-1 Logic modules of AB-IDPS

5.2 Aspect of HTTP Servlet

As introduced in the background section, an HTTP request from a web client will be converted into an HttpServletRequest object by the web application server. The HttpServletRequest object will be processed by the web application and then a HttpServletResponse object will be generated by the application. The HttpServletResponse object contains the response data required by the web client. The object will be converted into a regular HTTP response and sent back to the web client. As described earlier, the only way a web client can communicate with a web application is to

send HTTP requests to the web application server and get HTTP responses from it. All of the information data are transmitted through the HTTP layer in the TCP/IP model.

One of the most common approaches to detect and prevent web attacks is to analyze and filter HTTP conversations. It is used in most web application firewalls (WAF) in the world [17]. We also adopt this approach for implementing the AOP module in the prototype of AB-IDPS. The programming language that we use to develop the AOP module is called AspectJ.

5.2.1 HttpServletAspect

The file extension of a Java source file and class file are .java and .class, respectively. Similarly, the file extension of an AspectJ's class file is .class, but the source file extension is .aj. The file name of the HTTP Servlet aspect is called HttpServletAspect.aj, which is the core of the AOP module in this implementation.

```
public aspect HttpServletAspect
```

5.2.2 Pointcut

"A pointcut is a program element that picks out join points and exposes data from the execution context of those join points. Pointcuts are used primarily by advice. They can be composed with boolean operators to build up other pointcuts [13]." We define a composed public pointcut named servletRequestExec in the HttpServletAspect.

```
public pointcut servletRequestExec() :
    within(HttpServlet+) && call(* HttpServlet.do*(..));
```

We combine two pointcuts, within and call, using the boolean operator &&. First, it will identify each join point where the executing code is defined in a type matched by HttpServlet+. In other words, any join point inside the lexical scope of the HttpServlet

class and its subclasses will be picked out. Second, it will identify each method call join point whose signature matches HttpServlet.do* regardless of the return type and arguments. The goal is to identify each join point that is executing doGet and doPost method within HttpServlet and its subclasses.

5.2.3 Advice

In AspectJ, an advice is a piece of code that runs at each join point picked out by its pointcut. It defines what to do at the join points. There are three kinds of advice: before advice, after advice, and around advice. Before advice runs prior to the join point, but after advice executes following the join point. Around advice is able to run in place of its join point and "bypass execution, continue the original execution, or cause execution with an altered context" [13, 33].

We define an around advice for executing at the pointcut servletRequestExec() as well as passing the context from the join point to the advice. As the code shows below, it has this() and args() pointcuts except servletRequestExec(). The pointcuts this() and args() are provided by AspectJ to collect the context from join points. Therefore, the current execution object "servlet" and its arguments "request" and "response" will be passed to the around advice. Those objects will be used later in the advice code body.

```
void around (Object servlet, Object request, Object response)
 : servletRequestExec() && this(servlet) && args(request, response)
 {
```

The core of AOP module is within the around advice code body. It contains two separated processes. One is to verify the HTTP request. In this process, a new response object containing the original HTTP response is created. The advice passes the original

HTTP request and new response to the autonomic module for intrusion detection. A RequestHandler object will be holding the analyzed result from the autonomic module. If the result indicates that the original HTTP request and response should not be passed to the web application, the advice is able to return rejection information to the web user. Otherwise, the original request and response will be handled over to the web application.

Another process is executed after the request and response are sent back from the web application. The main purpose is to analyze the HTTP response. If the result in the RequestHandler object requests to check the HTTP response, then the advice needs to pass the request and response to the autonomic module again. Otherwise, the request and response will be sent to the web user without any changes.

The work flow of the around advice is shown in the flow chart depicted in Figure 5-2.

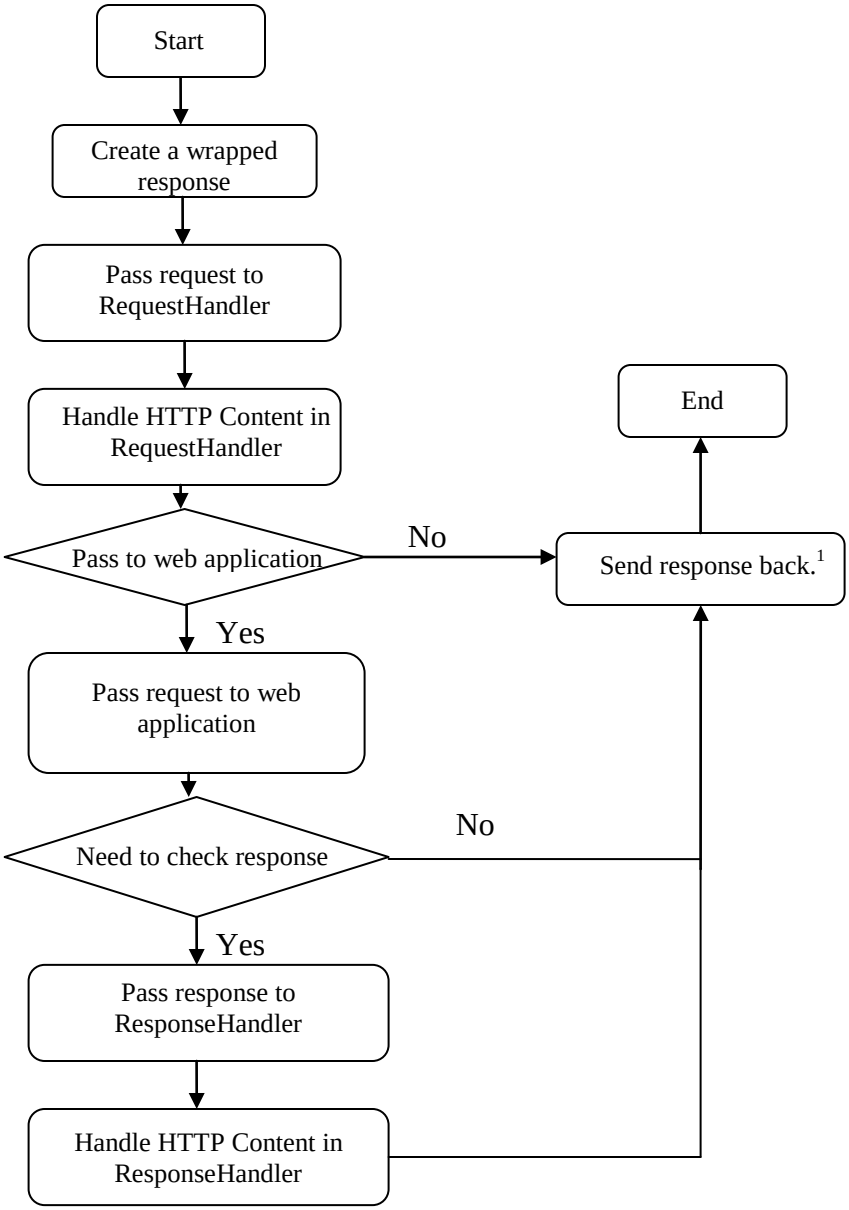


Figure 5-2 Flow chart of the around advice

1: The response may be modified by handlers if an intrusion occurs and the application has no ability to handle it.

5.2.4 Weaving with AspectJ at Load Time.

One of the biggest challenges for this implementation is how AB-IDPS is able to monitor and analyze any deployed application inside a Java web application server at

run-time. To achieve this, AB-IDPS must be loaded before any application was loaded. At the same time, the aspects to be used for weaving must be defined to the weaver before any types to be woven are loaded.

Apache Tomcat is an open source web container, which is the implementation of the Java Servlet and Javaserver Pages technologies [39]. In our experimental environment, we use Tomcat 5.5.26 as the web application server where the deployed applications are running. The load-time weaving mechanism is enabled through JVM start-up options, which are set in the start-up script under the Tomcat 5.5.26 bin folder. A configuration file, which needs to be created and put into AB-IDPS jar file, named aop.xml determines the set of aspects to be used for weaving and which types will be woven. The jar file must be put in the Tomcat's common lib folder where the jar files are loaded before any deployed applications. With these configurations, AB-IDPS is able to track all HTTP requests and responses transmitted through Tomcat.

5.2.5 HTTP Servlet Response Wrapper

As described in the last section, the AB-IDPS needs to get the content of the HTTP response that is being sent back to the web in order to analyze the contents to determine whether the web application has the ability to prevent the attacks. However, HttpServletResponseWrapper in the Java Servlet API package does not provide a getContent() method for developers to get the content of a HTTP response. We create our own response wrapper named WrapperResponse that extends the HttpServletResponseWrapper and overrides the getOutputStream() and getWriter() methods. In this way, the response contents will be written to the output stream in the WrapperResponse, instead of HttpServletResponseWrapper. A getContent() method is

created in WrapperResponse class to return the response content if it is called. The part of WrapperResponse code example is shown below.

```
public class WrapperResponse extends HttpServletResponseWrapper
{
    private PrintWriter tmpWriter;
    private ByteArrayOutputStream output;
    private ByteArrayServletOutputStream servletOutput;

    public WrapperResponse(HttpServletResponse httpServletResponse)
    {
        super(httpServletResponse);
        output = new ByteArrayOutputStream();
        tmpWriter = new PrintWriter(output);
        servletOutput = new ByteArrayServletOutputStream(output);
    }

    public String getContent()
    {
        try
        {
            String s = output.toString("UTF-8");
            return s;
        }
        catch(UnsupportedEncodingException e)
        {
            return "UnsupportedEncoding";
        }
    }

    @Override
    public PrintWriter getWriter() throws IOException
    {
        return tmpWriter;
    }

    @Override
    public ServletOutputStream getOutputStream() throws IOException
    {
        return servletOutput;
    }
```

Figure 5-3 HTTP Servlet Response Wrapper

5.3 HTTP Content Handlers

In the design of the prototype, the function of the autonomic module is to not only detect and prevent attacks within HTTP requests, but also check the response from the web application. In the implementation, a handler is an object that the system controller uses to connect the monitor, analyzer, and executor. There are two types of handlers: RequestHandler and ResponseHandler. They process HTTP requests and responses separately through autonomic elements.

5.3.1 Request Handler

The AOP module passes the HTTP request to RequestHandler in order to check the data in the request and then make decisions for subsequent executions. All executions are handled in the method handleHTTPRequest(). RequestHandler first creates a Request Monitor (RTM) for each HTTP request. The RTM decides whether to pass the original request over to the web application. The RTM will return a true value to the AOP module if the request is clear. Otherwise, the request will be passed to a Request Analyzer (RTA) for further investigation. The RTA processes the request in a similar way (i.e., deciding whether to pass the request to the web application). If the original request needs to be rejected due to intrusion prevention, a Request Executor (RTE) will be created to process the rejection. Otherwise, the request will be passed to the web application as usual. The code flow chart is depicted in Figure 5-4 below.

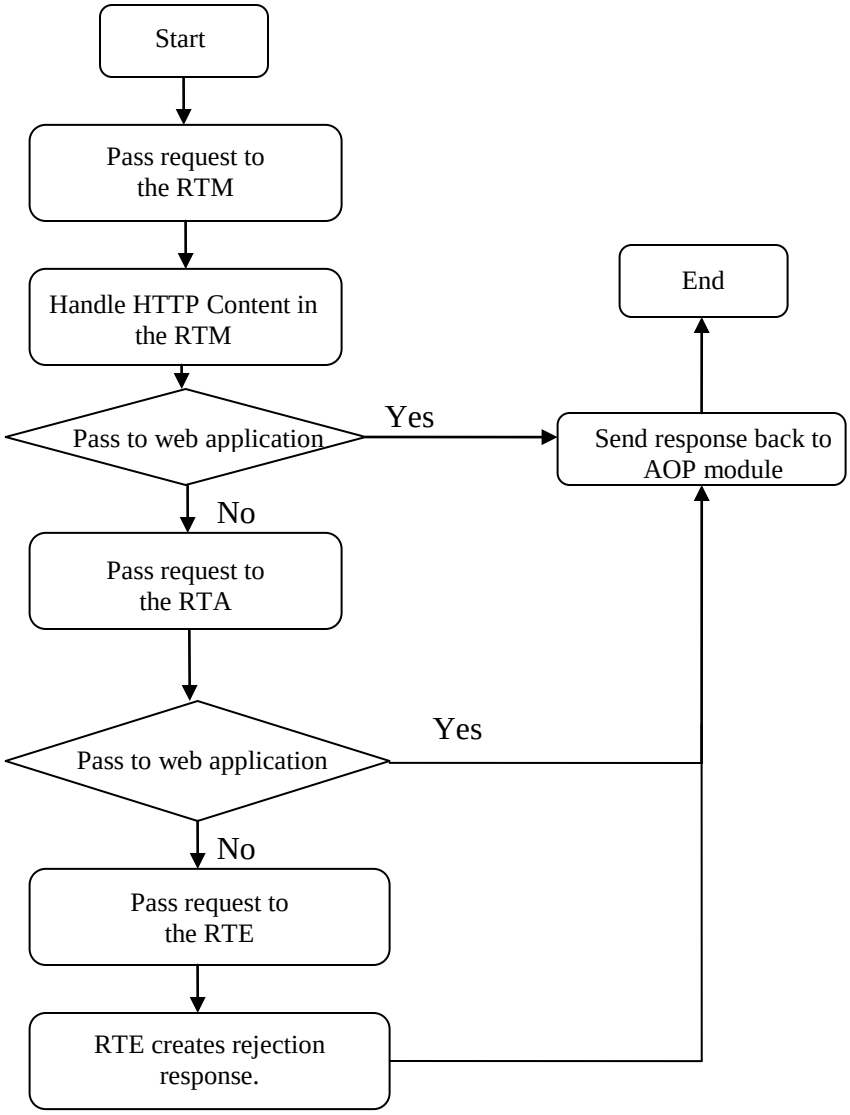


Figure 5-4 Flow chart of the handleHTTPRequest()

5.3.2 Response Handler

For our prototype, we developed a handler to analyze the data in the HTTP response. The tools or frameworks introduced in Chapter 3 focus on monitoring and analyzing HTTP requests, but are not intended to detect intrusions on the HTTP response from web applications. However, it is important to analyze the data in HTTP responses. First of all, the intrusion detection system should know whether the web application has

the ability to protect itself regarding selected types of attacks. This feature is included in the implementation. Second of all, the system should issue and record a warning regarding a possible intrusion if sensitive information is included in the HTTP response and it should not be returned. Most likely, an unknown type of attack has occurred in the HTTP request, but the current intrusion detection system is not able to discover it. This feature is not implemented in our prototype and left for future work.

The AOP module will pass the HTTP response that has been processed by the web application to the Response Handler if the intrusion to this application never occurred before. In other words, AB-IDPS needs to know whether the web application has prevented this kind of intrusion before or whether it is the first time. A Response Monitor (RSM) is created to check whether the response from the application is abnormal through the method handleHTTPContent(). After that, the response is passed to a Response Analyzer (RSA) for further validation. The RSA records the validation result into the knowledge base. This result indicates whether the web application is able to prevent the attack or not. This way, the knowledge base is acquiring more knowledge about the web application capabilities.

To avoid blocking legitimate responses, RSA is responsible for doing deep analysis to the responses from web applications. For example, RSA is able to detect whether scripts in an HTTP response are XSS or a normal script. An XSS is usually embedded in an HTTP request and sent back with the HTTP response to the client without any changes. Thus, RSA can compare the script in the HTTP response with the one in HTTP request. If both scripts contain the same content, then XSS web attack occurred.

A Response Executor (RSE) is created to deliver the rejection to the AOP module. When this attack occurs a second time, the AB-IDPS will not try to make the rejection response and let the application handle the attack if it is able to prevent the attack. The code flow chart is shown in Figure 5-5 below.

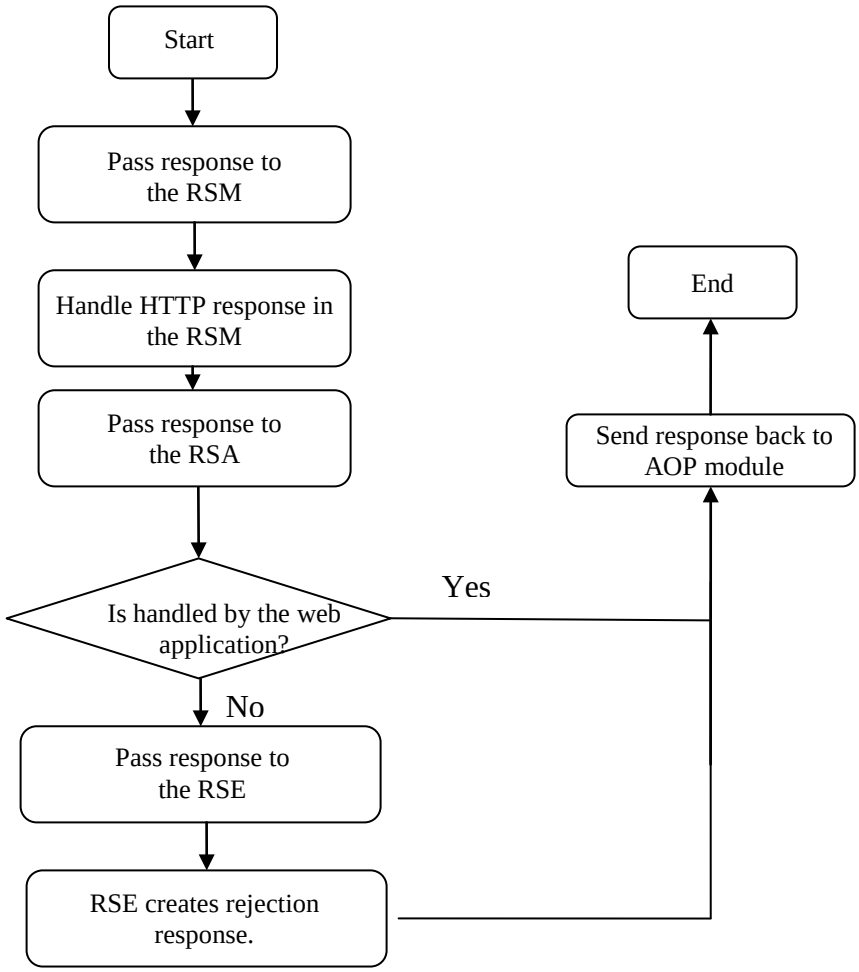


Figure 5-5 Flow chart of the handleHTTPContent()

5.4 Autonomic elements

As mentioned in the description of the system architecture, two autonomic elements were developed for this prototype implementation. One is in charge of detecting

and preventing the intrusion in HTTP requests, while another is responsible for HTTP responses. In the blueprint of autonomic computing architecture [32], each element has four functions: monitor, analyze, plan, and execute. In this prototype, we implemented different Java classes to realize these functions except for the planning function. In this implementation, we currently only consider intrusion detection and prevention for the known web attacks, such as XSS and SQL injection. The function plan is not necessary at this point because the rejection decision can be made immediately. However, it is the key feature of an intrusion detection and prevention system when detecting and preventing unknown intrusions based on the analysis of a large number of events or logs. The planning functionality should be well designed and implemented since the system needs to observe the behaviors of web applications over long periods of time and then make effective decisions. This is reserved for future work.

The knowledge base plays an important role in the prototype framework. The implementation of the knowledge base includes two sub-modules: dynamic and static sub-module. The dynamic knowledge base is mainly used by the monitors, while the static knowledge base is used by the analyzers in most cases. The dynamic knowledge base is able to determine whether there is an intrusion in the HTTP request. It could be a set of rules or event strings that are stored in a database or a file. Those rules or strings are created according to the known intrusion types or existing web attack approaches. For example, a rule to detect SQL injections may contain a string with "--" and single quote. These characters are widely used for SQL Injections.

On the other hand, the static knowledge base is composed of the data stored in a database. In the database, it keeps the records of the URL pages, the intrusion types and

their relationship. At run-time, the request analyzer (RTA) can execute a query to check whether the web application has an ability to prevent a certain type of attacks. The response analyzer (RSA) is responsible for creating and updating these records. The knowledge is passed by the response monitor (RSM) through validating the response from the web application. In the implementation, the monitors and analyzers communicate with the database through a class called DataBaseManager and its inner class UpdateDBThread.

The UML diagram depicted in Figure 5-6 shows the relationships of monitor, analyzer, executor, dynamic and static knowledge bases. The abstract classes are used for the high level design of the framework implementation. We could add several interfaces to make the design more flexible.

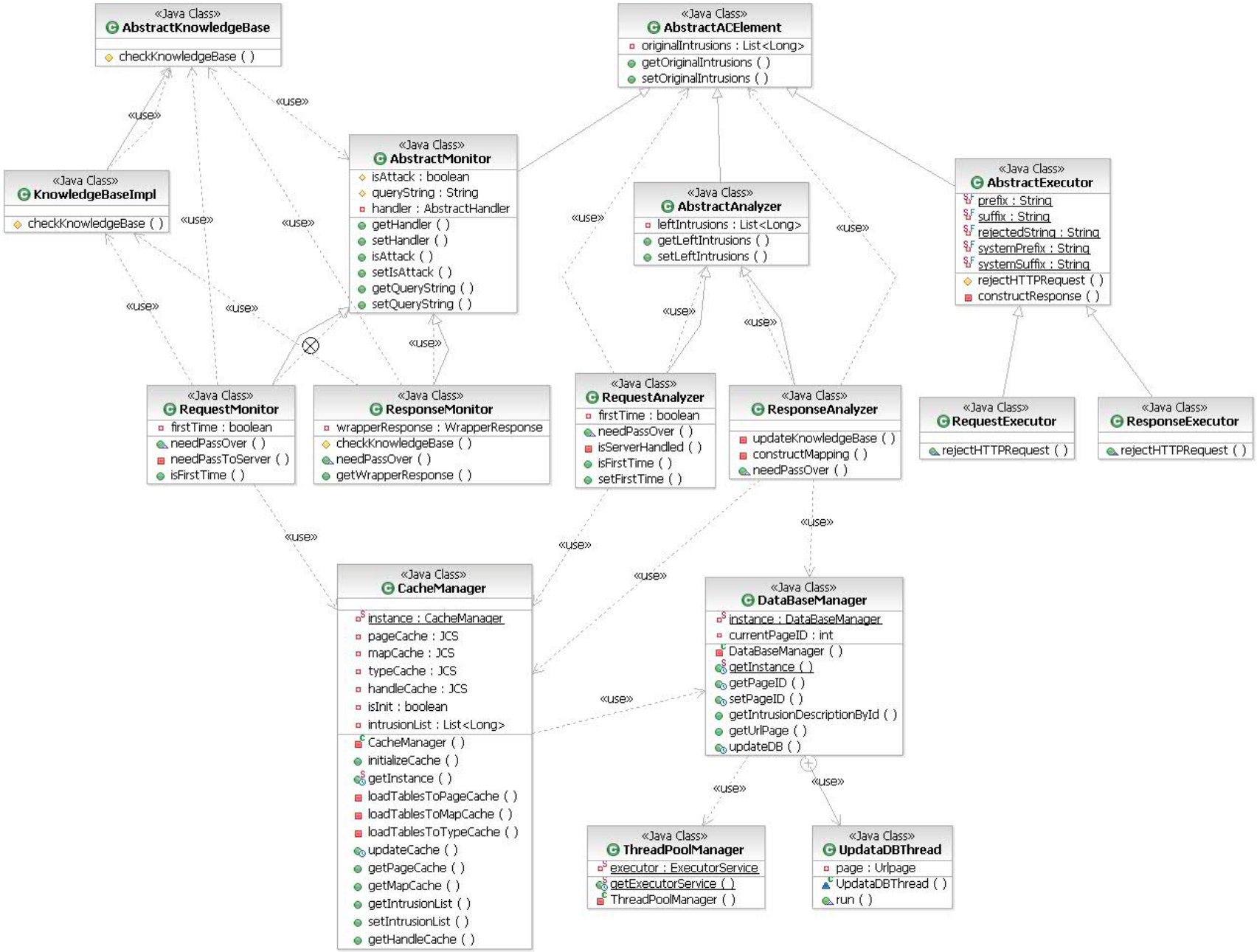


Figure 5-6 UML diagram of the Autonomic Elements

5.5 Knowledge Base

Knowledge Base is the most important module in both the design and implementation of our approach. As we mentioned before, the implementation of Knowledge Base is composed of static and dynamic sub-modules. Static sub-module stores all data in a database through the Java object DatabaseManager, and integrates a Java Caching System to improve the performance of reading data from the database. For comparison purposes, we use two different technologies for implementing the dynamic sub- module. The technologies are Rule Engine and Complex Event Processing (CEP).

A rule is like the if-then statements of programming languages and applies in certain situations. A rule engine is an environment to develop rules and then deploy them in the environment [35]. Complex Event Processing (CEP) is a defined set of tools and techniques for analyzing and controlling the complex series of interrelated events that drive modern distributed information systems [36]. CEP technology helps IT professionals effectively utilize events for enhanced system operation, performance, and security as well as quickly identify and solve problems with the system. CEP can be applied to intrusion detection, which is one of the most challenged areas of information system [36].

5.5.1 Drools Expert

Drools Expert is a Rule Engine that uses the rule-based approach to implement a business rule management system (BRMS) [37]. The main function of the knowledge base is to detect whether HTTP requests contain tainted data. Technically, these tainted data are strings that contain specific characters or match certain patterns. For example, we define a rule that indicates if the data contains a string “script”, then it will result in

attacks. There will be hundreds of these kinds of rules that are defined in AB-IDPS for intrusion detection. Furthermore, these rules need be assessed at the same time in a real time web based environment. The Rule Engine Drools Expert perfectly satisfies this requirement. It has well developed APIs for external programs to integrate seamlessly.

In this implementation, we only defined a few of rules using Drools. These rules are used to detect Cross Site Scripting and some types of SQL Injection. Drools Expert has its own rule language similar to other programming languages. A rule is defined using the rule language and stored in a rule file, which is typically a file with a .drl extension. Like other programming languages, the rule language is easy for developer to understand and use. Figure 5-7 shows an example rule that AB-IDPS is using to detect the XSS attack against HTTP requests.

```
rule "Request XSS Rule"

    no-loop
    when
        $rqm : RequestMonitor();
        eval($rqm.getQueryString().contains("<script")
                || $rqm.getQueryString().contains("javascript")
                || $rqm.getQueryString().contains("vbscript"));
    then
        System.out.println( "It is a XSS script in the http request." );
        $rqm.setIsAttack(true);
        $rqm.getOriginalIntrusions().add(1L);

        update($rqm);

end
```

Figure 5-7 Example rule to detect XSS attacks

The Java class RuleBaseImple has been implemented to initialize Drools Rule Engine when the system starts. All of the predefined rules in the rule files are loaded into the Drools' working memory. At run-time, a Request Monitor (RTM) passes itself into

working memory, and Drools will execute the code in those rule files. For example, like code in Figure 5-7, the HTTP query string will be extracted by using getQueryString() and evaluated to determine whether it contains string “<script”, “javascript”, or “vbscript”. If these strings are found in the HTTP query string, the code invokes setIsAttack() with true value and updates the RTM. In this way, the RTM gets the detection result from the rule engine. It is able to make next movement in terms of this result.

5.5.2 Esper Engine

Esper is a Complex Event Processing (CEP) Engine that works *like a database, but upside-down*. A normal relational database stores the data and runs queries against stored data. In contrast, the Esper engine allows applications to store queries and run the data through. Esper provides its own language called Event Processing Language (EPL), which “allows expressing rich event conditions, correlation, possibly spanning time windows, thus minimizing the development effort required to set up a system that can react to complex situations [38].” Since Esper is written in Java, it is able to fully embed into any Java program, process, JEE application server, or Java-based Enterprise Service Bus. Developers can rapidly develop applications that process large volumes of incoming messages or events [38].

In this prototype framework, we integrated Esper Engine with the autonomic module as a part of the knowledge base. We developed an adaptor called EsperAdaptor to connect to Esper Engine. The engine is initialized when the system starts. The query statements written in EPL are added into the engine instance. Two listeners are

instantiated and inserted into the engine to monitor the events, which are the object of Request or Response Monitor. The method update() in the abstract class BaseListener will be executed if the events satisfy the query statements. The code example and UML diagram are shown in Figure 5-8 below.

```
public abstract class BaseListener implements UpdateListener
{

    public void update(EventBean[] newEvents, EventBean[] oldEvents)
    {
        System.out.println("Using Esper Event Stream Processing : \n\t" + newEvents[0].toString());
        AbstractMonitor monitor = (AbstractMonitor) newEvents[0].getUnderlying();
        monitor.setIsAttack(true);
        monitor.getOriginalIntrusions().add(ABIDSUtils.getIntrusionType(monitor));

    }
```

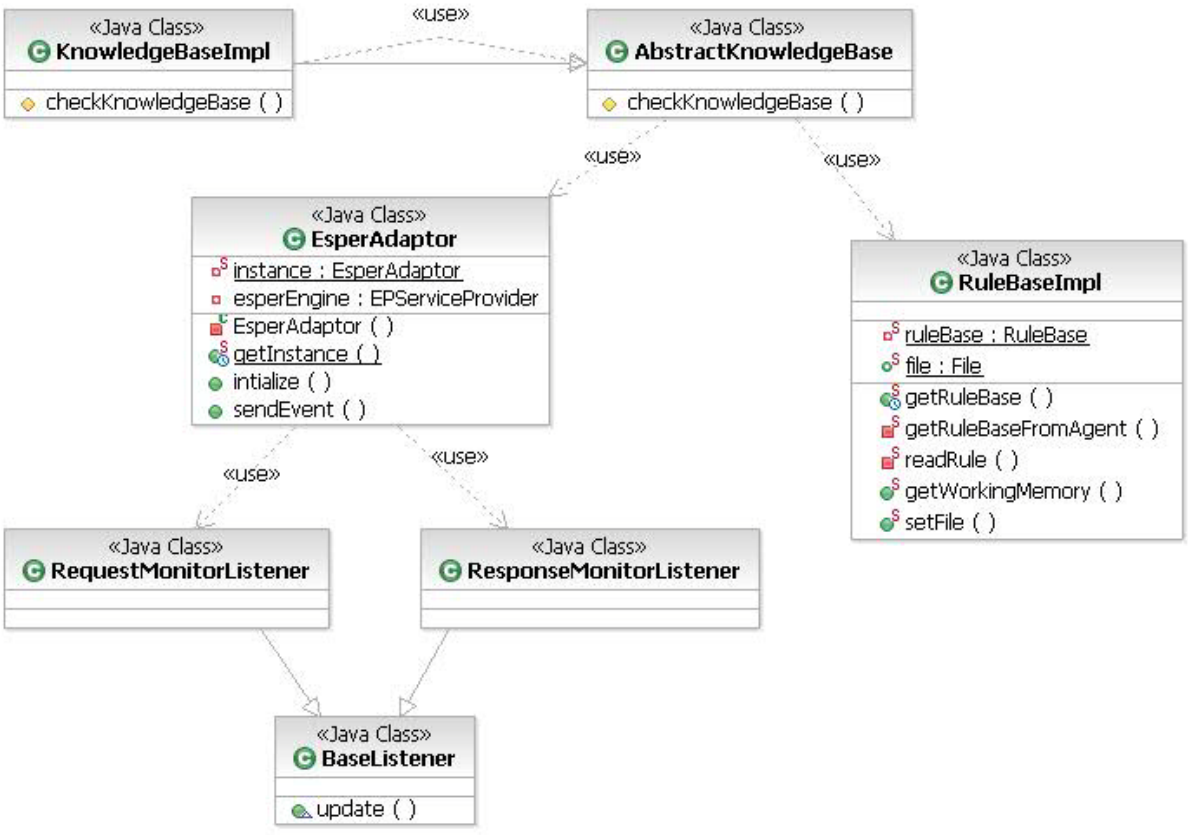


Figure 5-8 UML diagram of the knowledge base

In the following code, we use EPL to write a query for Cross Site Scripting detection. The query string is very similar with the rules we define in the rule file of Drool Expert.

```
// Get an engine instance
esperEngine = EPServiceProviderManager.getDefaultProvider(config);

String whereString = "where queryString like '%script%' " +
                     "or queryString like '%javascript%' " +
                     "or queryString like'%vbscript%'";

String stmt = "select * from RequestMonitor " +  whereString;
EPStatement statement = esperEngine.getEPAdministrator().createEPL(stmt);
statement.addListener(new RequestMonitorListener());
```

The code example below is for detecting SQL Injection Tautology. The query will select all events of RequestMonitor that have a Boolean variable *isTautology*, which equals *true*. The value of *isTautology* is set in the RequstMonitor if the WHERE clause is evaluated to be always *true*.

```
// Get an engine instance
esperEngine = EPServiceProviderManager.getDefaultProvider(config);

String stmt = "select * from RequestMonitor where isTautology = true";
EPStatement statement = esperEngine.getEPAdministrator().createEPL(stmt);
statement.addListener(new RequestMonitorListener());
```

5.6 Data Caching Manager

The static sub-module maintains the mapping of URL links and intrusion types in a database. The database tables that store those data may be getting larger and larger when the system is running for protecting a number of web applications. At run-time, AB-IDPS frequently search static data in the database to check whether a web page is vulnerable to some intrusion types. The large number of read operations hit the database, and it may result in a performance problem. One solution is to develop a caching system

that loads the data from the database when the system is initialized at the beginning.

In the implementation, we developed a class named CacheManager to integrate Apache Java Cache System (JCS) into the system. JCS is a distributed caching system built on Java platform. It is designed in an attempt to improve the efficiency of applications by providing a means to manage cached data of various dynamic natures. Like any caching system, JCS is most useful for high-get and low-put applications. The employment of JCS will leads to sharply dropped latency times and the remove of bottlenecks from the database in an effectively cached system [34].

The caching system is initialized with the knowledge base module when AB-IDPS first starts. CacheManger creates JCS objects for each table in the database, loads the data from database tables, and put them into the caching system. CacheManager object is able to maintain these caching objects, and return them when the monitors or analyzers need to check the mappings of URL and intrusion type. Also, CacheManager updates the caching system at run-time if a new piece of mapping data is added into the database. In this way, the caching system always has the latest version of intrusion mapping data in the memory. Reading these data from memory is much faster than from the database directly. The performance of the AB-IDPS is much improved, and the response time to a HTTP request drops significantly. The relationship of the autonomic elements and CacheManger is shown in Figure 5-6.

5.7 Summary

In this chapter we introduced the prototype tool of the AB-IDPS framework. Three key modules and two helper modules work together to demonstrate how the AB-

IDPS framework can be applied to detect and prevent common attacks, such as Cross Site Scripting and SQL Injection. This tool is built on Java and can be deployed to any Java web containers or web application servers. We evaluate this prototype tool by applying it to protect two web applications in the next chapter.

.

Chapter 6 Evaluation

This chapter evaluates the prototype tool by experimenting with it using two web applications, which consist of the presentation layer, the business layer, and the data storage layer. They can be deployed on a web application server such as JBoss or a web container such as Apache Tomcat. We used Apache Tomcat 5.5.26 for evaluating our prototype tool and used PostgreSQL Server 8.2 [41] as the database manager. The tool has abilities to detect and prevent Cross Site Scripting (XSS) and SQL Injection at the same time for both web applications. Based on our experiments, the AB-IDPS framework will be applicable and generalizable to any Java web applications, which can run on a Tomcat Web Container, for intrusion detection and prevention.

We did not fully implement our framework in this prototype tool, thus evaluating this prototype tool is just the first step for assessing our framework design. The current evaluation verifies whether the framework design using AOP and AC technologies is valid for detecting and preventing XSS and SQL Injection. For further evaluations, such as performance testing, new rule engines need be plugged in and different comparison approaches comparison need be developed.

6.1 Deployment and Integration

The framework prototype is easy to deploy on a web server or container such as Tomcat 5.5.26. The prototype source code can be complied and packed into a jar file. In our implementation, the jar file is called AB-IDPS.jar. The jar file is stored in the Tomcat common library folder, which is TOMCAT_HOME/common/lib. Also, we need to place aspectjweaver.jar in the same folder. The aspectjweaver.jar contains the feature of Load-

time weaving (LTW) of AspectJ. We can use Java 5's *javaagent* start-up parameter to invoke AspectJ LTW. To use this feature, we simply add a new Java option flag for Tomcat start up which is achieved by editing a shell script. For example, we could add the following line to the start of setclasspath.bat in a Windows environment:

```
set JAVA_OPTS=-javaagent:%TOMCAT_HOME%\common\lib\aspectjweaver.jar
```

We change the setclasspath.sh script similarly for Linux or other Unix operating systems. This setup simply allows for load-time weaving in the Tomcat VM. The AspectJ weaver finds the aop.xml file in AB-IDPS.jar file.

Thanks to AspectJ LTW, the prototype's AOP module can be weaved into the web applications loaded by Tomcat VM. The aspect of HttpServlet is applied to the abstract class of HTTP Servlet directly. Therefore, the AB-IDPS prototype is not related to any web applications running in the Tomcat container. Thus, it is fairly easy to integrate the AB-IDPS prototype with a Tomcat container and its web applications.

We did not apply our framework prototype to other web servers or containers (e.g., Oracle iPlanet Web Server). The web applications running on other web servers or containers might not be integrated with our implementation successfully.

6.2 Overview of Experimentation Setup

For our experiments, we modified two existing web applications to be vulnerable to SQL Injection and XSS attacks. In the beginning of the experimentation, we tried all types of SQL Injection and XSS attacks introduced in the background chapter, to see how these applications behaved. Then we protected them with AB-IDPS using Drools Expert and Esper. We attacked the application again, but the attacks were unable to bypass the applications' security. Any HTTP requests embedded with any kind of scripting language,

such as JavaScript or VBScript, were detected by AB-IDPS. Similarly, AB-IDPS was able to detect two types of SQL Injection, such as tautologies and comments, contained in an HTTP request.

For example, an attacker tries to input the following HTTP request through the login web page in order to obtain information as a system administrator:

http://www.web.com/daytrading/processLogin.do?username=admin'--&password=''

The actual query is as follows:

*select * from users where username='admin' --' and password=' ';*

The request will not be processed because it contains a comment inside it. The AB-IDPS will detect it and refuse to pass it to the web application.

In another example, the attacker will try to obtain information using an HTTP request that contains a statement that is always true.

http://www.web.com/daytrading/processLogin.do?username=admin&password=' or 1=1

The actual query looks like this:

*select * from users where username='admin' and password='' or 1=1;*

The AB-IDPS will detect that the request contains a statement that is always true and will reject processing it.

The results show that AB-IDPS is able to protect Java web applications from any types of XSS attacks and all types of SQL Injection described in the background chapter. The following two sections describe the experiments in detail for the DayTrading System and Online Photo Store.

6.3 DayTrading System

DayTrading System is a stock trading system that handles the customer stock operations such as buying and selling stocks on-line. It contains not only a web application, but also many software programs and hardware systems (e.g., firewall, load balancer, server cluster, and caching systems). The customers buy or sell stock through the web user interface and all transaction data are stored in a database. Each customer has its own account and has to be authorized to log on to the account. The system architecture is designed in terms of the business model and the required features of a stock trading company. Based on analysis of the requirement and existing industry standards, the system architecture is divided into several subsystems pertaining to distinct functionalities. The system architecture is depicted in Figure 6-1 below.

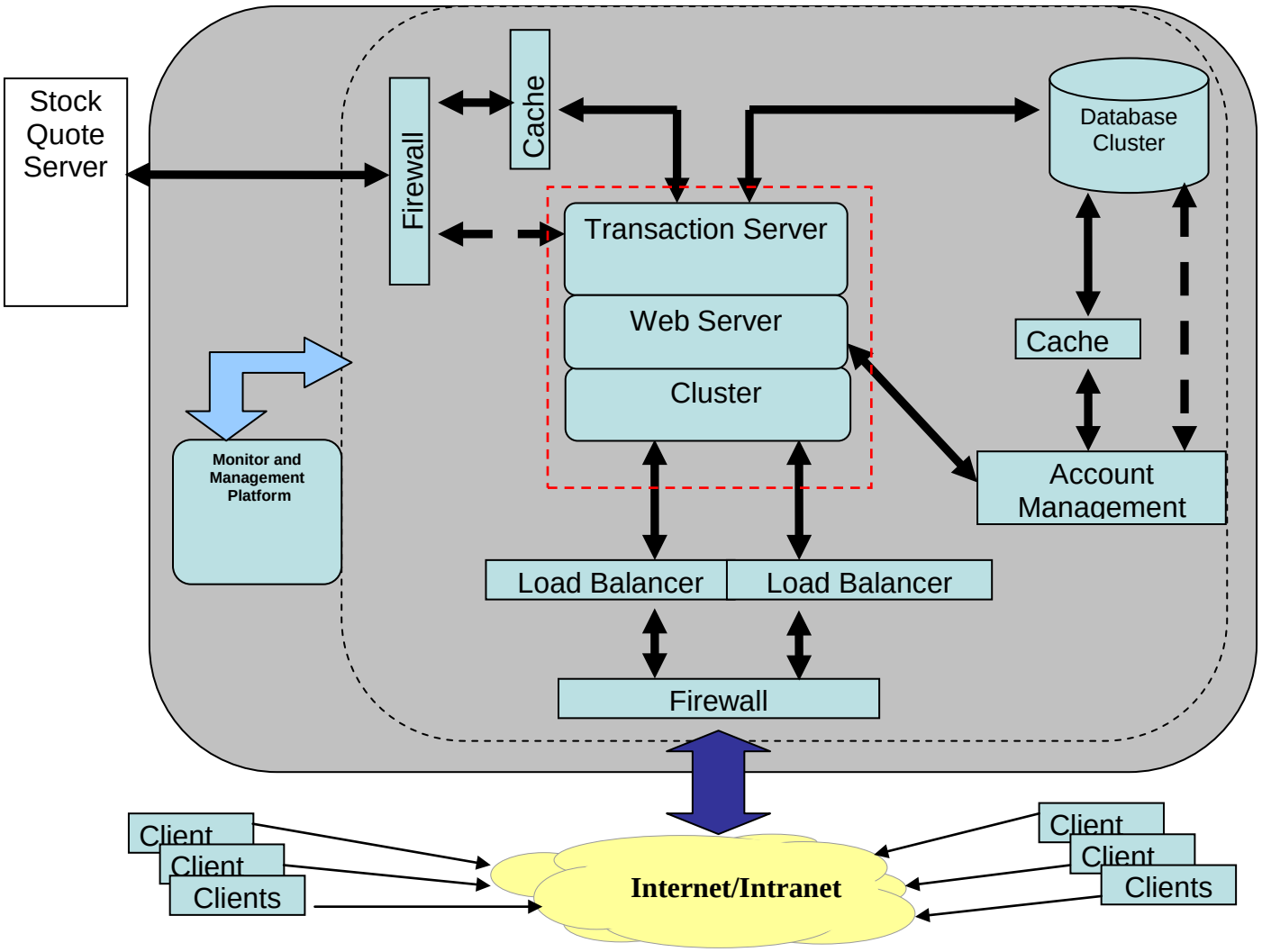


Figure 6-1 DayTrading System architecture

Even though two firewalls have been used to protect the system from system-based intrusions, many web-based attacks could still bypass the security system. AB-IDPS is able to protect DayTrading System from web-based attacks. AB-IDPS and the web application of DayTrading System must be deployed in the same web application container, which isTomcat 5.5.26 in our experimental environment.

6.3.1 Cross Site Scripting Attack

Like most web applications, DayTrading web application has a web page that provides users to log on to the account using username and password. For example, if a user types the username and password correctly on the logon web page, then the system will allow the user to enter his or her account information as shown in Figure 6-2. However, the system does not filter some special characters such as "<script>" from the input of the username field, but returns the input back to the web browser without validation. This vulnerability provides hackers a chance to use Cross Site Scripting Attack.

Figure 6-2 DayTrading user logon web page

We can enable or disable the function of intrusion detection and prevention when performing the experiment. In order to show that the vulnerability exists in DayTrading web application, we first disable the function by changing a variable in the AB-IDPS configuration file. In the configuration file, there is another variable that is used to enable Drools Rule Engine or Esper Engine. The configuration file is shown in Figure 6-3 below.

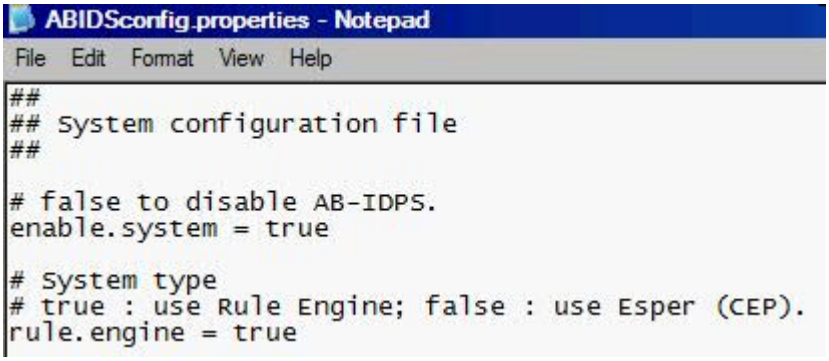


Figure 6-3 AB-IDPS configuration file

6.3.1.1 AB-IDPS Disabled

The following screenshot show a user logs on to the account normally.

Figure 6-2 DayTrading user logs on successfully

If the username or the password is input incorrectly, an error page is shown as depicted below. As we can see, the username “Qin Zhu” is returned and shown on the web page. This implies that someone could use Cross Site Scripting to attack this web application.

Figure 6-3 DayTrading username or password is incorrect

A simple XSS attack example is to input the string “<script>alert("XSS Test")</script>” in the username field with a random password.

Figure 6-4 DayTrading XSS attack example

The following screenshot shows how the attack succeeds. The Java Script is run successfully through the logon web page on this web site.

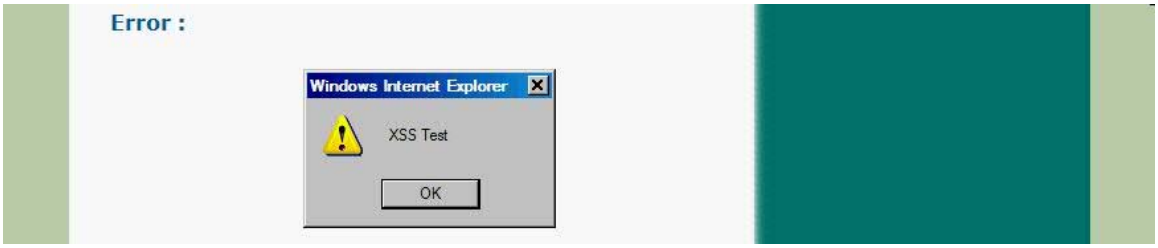


Figure 6-5 DayTrading XSS attack succeeds

This experiment proves that more harmful scripts could be run successfully through the use logon web page. For instance, we can input "<script>alert(document.cookie)</script>" in the username field and execute it to show the cookie of the current HTTP session. Hackers could write more sophisticated scripts to steal information silently and secretly.

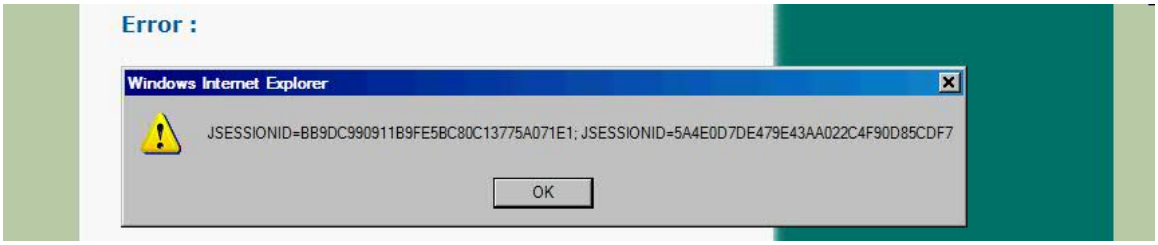


Figure 6-6 DayTrading user's cookie is shown after the attack

6.3.1.2 AB-IDPS enabled

To enable intrusion detection and prevention, we only need to set the value of "enable.system" to true and restart Tomcat. On the user logon web page, we input the XSS attack strings used in the last section. The AB-IDPS detects the XSS attacks and learns that DayTrading web application is not able to protect itself against these attacks. Rejected information is sent back to the web browser as shown below.

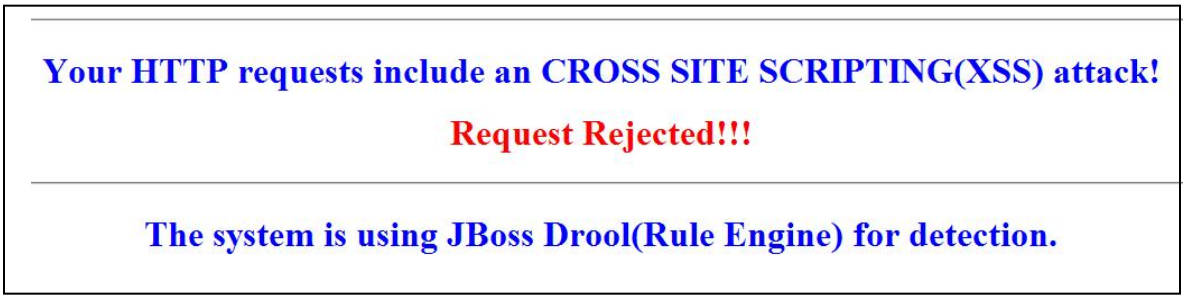


Figure 6-7 XSS rejected information from AB-IDPS using Drools Expert

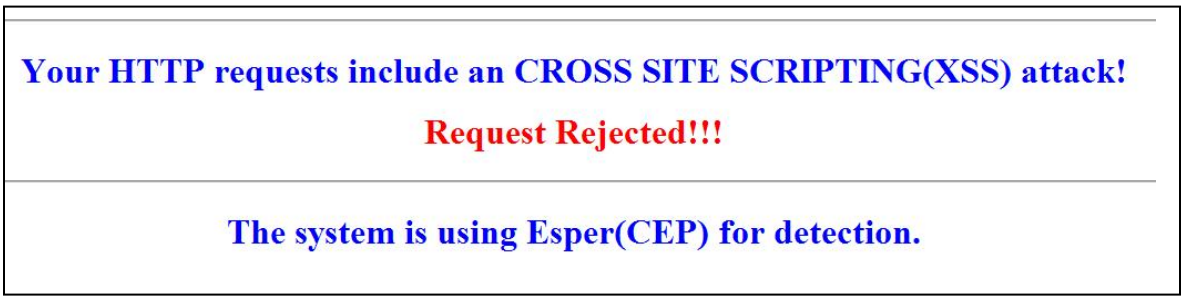


Figure 6-8 XSS rejected information from AB-IDPS using Esper Engine

6.3.2 SQL Injection Attack

To demonstrate an SQL Injection attack, we first disable intrusion detection and prevention to show this vulnerability of DayTrading web application.

6.3.2.1 AB-IDPS Disabled

The typical SQL Injection example is to use a string containing "' or 1=1 --". As described in the background chapter, a query which contains this kind of string could be interpreted to be always true. As a result, the logon action can be successful and the account returned usually is the first user in the database table. The user who is hacked could be "admin", "administrator", or others, which is the first user in the database table.

Figure 6-9 DayTrading SQL Injection example

6.3.2.2 AB-IDPS Enabled

AB-IDPS will detect and prevent the SQL Injection attacks if we enable it in the configuration file and restart the Tomcat web container. The AB-IDPS detects the SQL Injection attacks and learns that DayTrading web application is not able to protect itself against these attacks. Rejected information is sent back to the web browser as shown below.

Figure 6-10 SQL Injection rejected information from AB-IDPS using Drools Expert

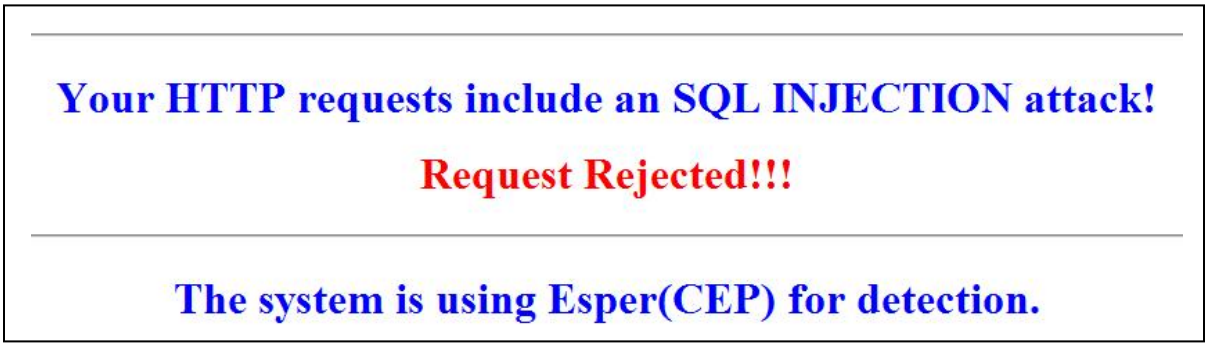


Figure 6-11 SQL Injection rejected information from AB-IDPS using Esper Engine

6.4 Online Photo Store

In contrast to DayTrading System, Online Photo Store is a simple web application with a basic structure. Users can purchase photos and check their history purchase on-line. It has a simple web user interface, a business model to handle purchase transactions, and

a database to store authorization and purchase history data. The user has to be authorized before logging on to his or her account. The system uses MySQL Community Server 5.1 [42] as the database manager.

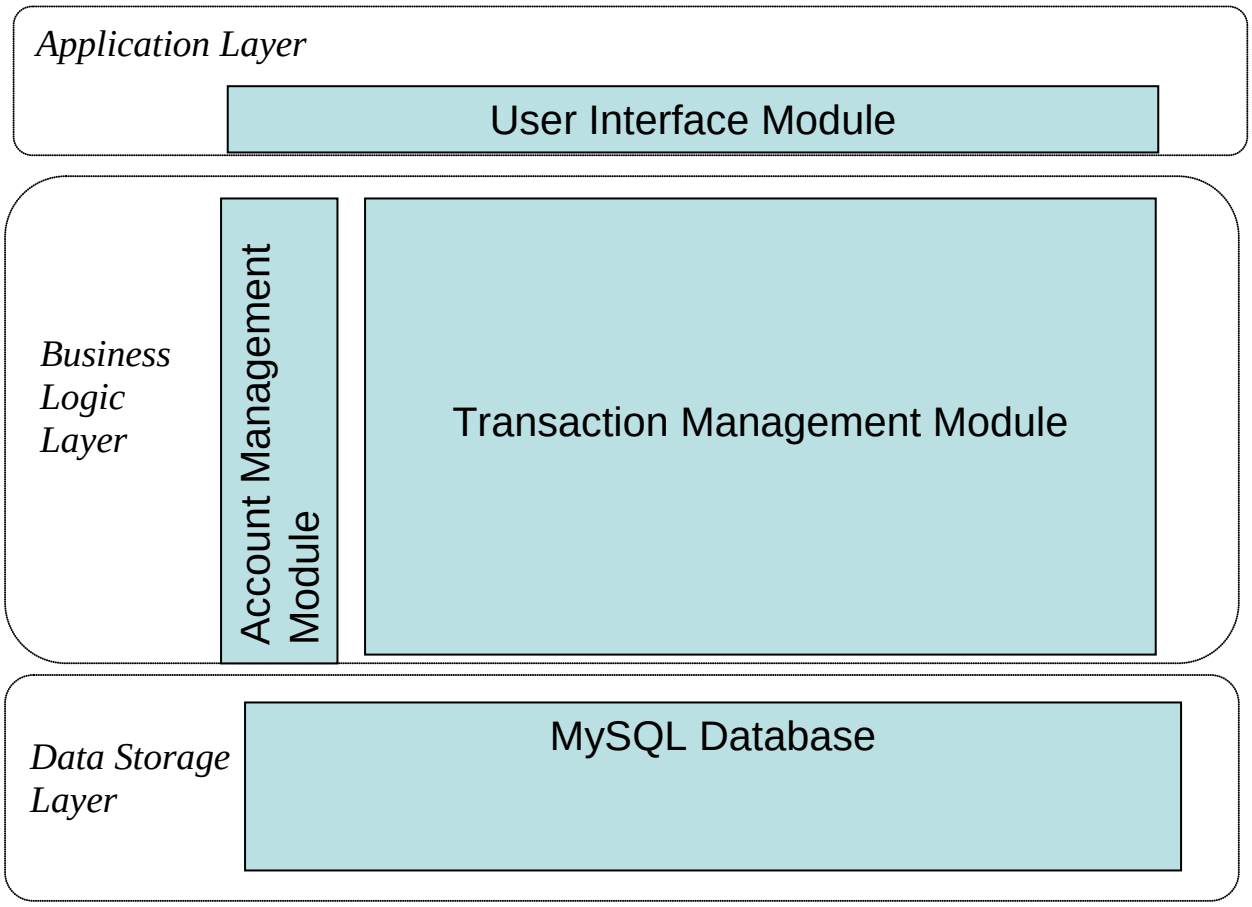


Figure 6-12 Online Photo Store system architecture

6.4.1 Cross Site Scripting Attack

Cross Site Scripting can also be used to attack Online Photo Store from the user logon web pages.

6.4.1.1 AB-IDPS Disabled

AB-IDPS is disabled for this experiment to illustrate the vulnerability of Online Photo Store. The following screenshots show an attempt of a XSS attack by inputting the string "<script>alert(document.cookie)</script>".

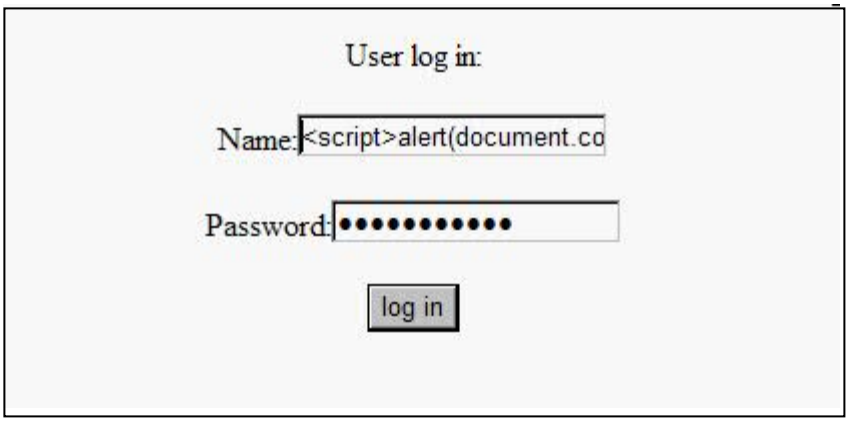


Figure 6-13 Photo Store XSS attack example

The cookie is shown after the script is executed on the web page.

Figure 6-14 Photo Store user's cookie is shown after the attack occurred

6.4.1.2 AB-IDPS Enabled

The attack is prevented if AB-IDPS is enabled. The same rejected information is shown as Figures 6-8 and 6-9. The HTTP session is terminated when the response is sent back to the web browser.

6.4.2 SQL Injection Attack

The same attack string used on DayTrading System can also be used to attack Online Photo Store.

6.4.2.1 AB-IDPS Enabled

The string ' or 1=1 --' is typed in the username field while a random password is inputted in the password field.

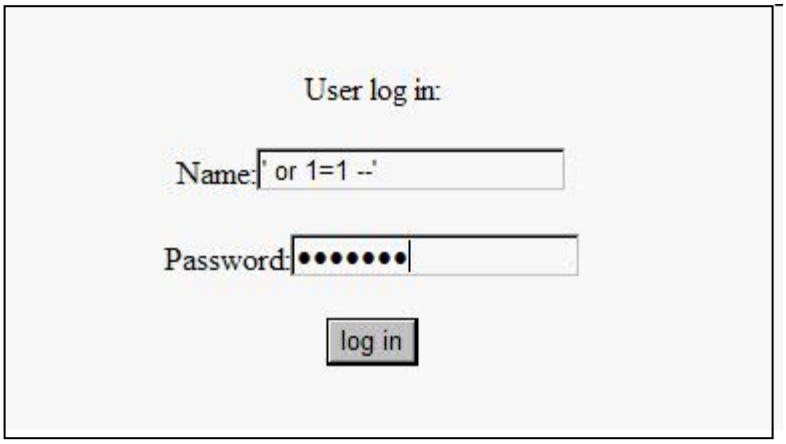


Figure 6-15 Photo Store SQL Injection attack example

The following screenshot shows that the attack is successful. The account name is ' or 1=1 --', but it actually is the first account in the database users table.

Figure 6-16 Photo Store SQL Injection attack succeeds

6.4.2.2 AB-IDPS Enabled

Similar to the XSS attack experiment, the SQL Injection attack is prevented if AB-IDPS is enabled. The same rejected information is shown as depicted in Figure 6-11 and 6-12. The HTTP session is terminated when the response is sent back to the web browser.

6.5 Development Experience

The initial design for the prototype is to implement a dynamic knowledge base sub-module using Drools Expert API. However, this thesis is part of the collaboration research project, and the implementation of AB-IDPS framework is part of the research platform. The purpose of building this platform is to demonstrate ideas and perform experiments. Using Complex Event Processing (CEP) technologies to diagnose intrusions is one of the experiments conducted for this research project. Therefore, we implemented another dynamic knowledge base sub-module using Esper Engine, which is an implementation of CEP for Java. It is also an opportunity for us to compare these two libraries with respect to development experience.

Drools Expert is an expert system written in Java with rule-based techniques. It uses defined rules to perform pattern matching on different conditions [37]. The rules can be defined in a source file using a rule language with Java like syntax. The source file is known as a rule file written in plain text format. Thanks to this feature, the rule files are separated from the Drools Expert system and can be loaded dynamically at run time. In the implementation of AB-IDPS framework, we integrated Drools Expert into the prototype tool as part of Knowledge Base. The known intrusion detection methods can be

written in a rule file and loaded at system start-up time, while the newly developed functions can be added into the file easily and loaded at run-time. This way, the AB-IDPS does not need to be restarted when adding new features to detect and prevent newly discovered web attacks.

In contrast, Esper Engine is an implementation of Complex Event Processing (CEP) and also written in Java. It is able to process and analyze events or messages in real-time or near real-time. It can be applied to the applications requiring high throughput, low latency, and complex computations [38]. The Esper engine offers an event pattern language to define expression-based event pattern and event stream queries to handle the event stream analysis. Esper provides the windows, aggregation, joining and analysis functions for event streams. These functions can be used to monitor and record the malicious activities for further analysis. For example, AB-IDPS can use these Esper functions to monitor and record HTTP traffic from a malicious user within a time window. The analyzer can perform an investigation on the recorded data to determine whether a new type of intrusion occurred.

The event pattern language or event stream queries need to be embedded into the Java application code and compiled together. Unlike Drools Expert, Esper Engine does not provide an easy way to load some code at run-time. Generally, the event pattern language or event stream queries need to be compiled with the application code. Consequently, the Esper application developers have to design and implement a sophisticated mechanism to load new code dynamically at run-time. From an implementation perspective, this is a drawback for Esper Engine compared to Drools Expert.

Drool Expert and Esper Engine have different technology focuses, but they can be applied to achieve the same business goal, like intrusion detection and prevention. Drools Expert is suitable for detecting web attacks using predefined rules, while Esper Engine could be used to discover new web attacks by monitoring malicious activities. We integrated these two APIs together to implement the AB-IDPS prototype tool.

6.6 Limitations of the Prototype

In this thesis, we designed an IDPS framework using AOP and AC technologies. We also developed a prototype tool to implement this framework design. The prototype tool has limited capabilities since it is intended only for the experimental purposes. The limitations are as follows:

- Not able to detect all known types of SQL Injection. Only able to detect and prevent two types of SQL Injection: tautologies and additional comments.
- Not able to detect other known types of web attacks, such as those in the Top 10. Further development work is required.
- Not able to use Drools Expert and Esper at the same time for detection.
- Not able to detect new types of web attack from malicious activities.

Even though there are limitations of this implementation, AB-IDPS has been designed to be an open framework for researchers and developers. With further development work, we can extend the tool to detect and prevent other types of SQL injection and other web vulnerabilities, and even detect new types of vulnerabilities.

6.7 Research Collaboration

As mentioned in the introduction, this thesis is part of an industrial collaborative research project funded by CA. Canada Inc. The experimentation of AB-IDPS is used as a case study by Zhu in his Ph.D. dissertation [43]. Zhu discussed and explored several solutions to improve effectiveness and efficiency of the computer-aided Root Cause Analysis and Diagnosis (RCAD) process. The approach proposed by Zhu leverages CEP technology and makes the RCAD process autonomic through properly designed and implemented use-case-unit Event Processing Networks (EPNs).

The experimental environment described in this thesis provides substantial assistance to Zhu's research. The DayTrading System discussed in this thesis is used in Zhu's work as a research platform, and the prototype of AB-IDPS is used as an experimental tool. Based on this thesis, Zhu is able to build up his experiments and validate his approach. As a result, we both made important contributions to CA's collaborative research project.

6.8 Summary

In this chapter, the AB-IDPS prototype was evaluated against two web applications: DayTrading web application and Online Photo Store. We first showed how the web applications behave without enabling AB-IDPS. Both XSS and SQL Injection attacks were executed successfully. However, the attacks were detected and blocked after AB-IDPS was enabled. Due to the limitations, we could not experiment the prototype against more types of web-based attacks such as the Top 10 list. The improvements in this direction are left for future work.

Chapter 7 Conclusions

7.1 Summary

Web applications have become a wide-spread interaction medium in our modern society. Due to regularly discovered vulnerabilities of web applications, the detection and prevention of the vulnerability and intrusions are labour intensive, costly, and error-prone. In this thesis, we explored how Autonomic Computing and Aspect-Oriented Programming (AOP) technologies can be used to design and implement an intrusion detection and prevention system. An open framework (AB-IDPS) is proposed to be applied transparently to deployed web applications to increase their security.

Many techniques and concepts have been employed in AB-IDPS to achieve a flexible, maintainable, and scalable system. The AOP mechanism was applied to design an HTTP Servlets aspect, which performs the functions of a sensor and an effector. Two autonomic elements have been designed for managing HTTP requests and responses with a shared knowledge base. As the key component of the architecture, the knowledge base is designed as an open structure with static and dynamic modules. The static module is used to record or store the data regarding web applications' security abilities, while the dynamic module is to store rules or events used by rule engines. At the same time, the system can be easily extended by adding new components to the dynamic module.

A prototype tool is developed based on our framework design. Experiments are conducted by applying the prototype tool to two web applications. As shown in the experimental results, the tool is able to detect and prevent XSS attacks and two types of SQL injection. The comparison of two third-party libraries, Drools Expert and Esper

Engine, has shown that both have their own advantages. The limitation of the prototype tool has also been described in the evaluation chapter.

The design of the framework also provides means to enable the capacity of the prototype tool to be easily extended and efficiently executed in common intrusion detection and prevention. With additional new techniques, such as artificial intelligence technologies, plugged into the knowledge base, the prototype tool could be enhanced to detect and prevent new or unknown attacks.

We are always looking for more effective approaches to detect and prevent web-based attacks, especially with the rapid growth in the size and complexity of web applications. We believe that the presented framework provides a possible solution to the stated problem and offers many benefits to both users and web application providers.

7.2 Contributions

In this thesis, we discussed and investigated background information and related work regarding web attacks in an attempt to provide a general and an efficient way to protect web applications. Based on the study, we designed an intrusion detection and prevention system framework called AB-IDPS and developed a prototype tool based on this framework design.

The main contributions of this thesis are as follows:

- We conducted background research and identified related work;
- We proposed a framework of Intrusion Detection and Prevention System using Aspect-Oriented Programming diagram and Autonomic Computing technologies;

- We developed a tool prototype to assess this framework design;
- We compared Drools Expert with Esper Engine based on our development experience;
- We collaborated with Zhu on his dissertation research; and
- We contributed to the CA Canada Inc. research project.

7.3 Future work

There are many aspects of this research that can be further studied and explored. In this section, we discuss possible future work of this research and potential improvements for the tool prototype. Much research has been conducted for decades and many new intrusion detection technologies are emerging rapidly in this area. We feel that further investigations are necessary on how to detect new vulnerabilities of web applications more effectively. For example, Artificial Intelligence (AI) is one of the broadest research areas. Many technologies such as Machine Learning and Neural Networks could be applied to detect new types of web attacks,. We could employ these technologies to develop more sophisticated detection approaches based on event logs with long time observation. Consequently, the AB-IDPS framework could be extended to improve the process of intrusion detection by further analyzing inputs and outputs of a web application system, and thus discover and prevent new types of web attacks.

Several places of the tool prototype could be improved as well. In the process of developing the tool, we did not implement the Request and Response Planner completely. The planners were implemented with the analyzers since their functionalities can be combined together as a whole for detecting and preventing known web attacks. However, the planners will need more functionality and responsibilities if new techniques based on

AI technologies need to be introduced for the detection of new web attacks. For example, the planners could make a long term plan to collect data by monitoring the event stream and observing the system behaviours, and then make a decision based on these collected data. In this case, the planners are important and need to be implemented with many functions.

It would also be useful to further improve the usability of the prototype. In the tool, the Request and Response Executers only have a function to block the HTTP requests and send back rejection information. The current executers could be improved to present more capabilities. For instance, an executer could send an alarm or warning message to the system administrators once a web attack occurs. It could also suspend the user account if the account has been used to attack the system. In the presentation layer, the web-based user interface could be implemented to visualize the data in the knowledge base and display HTTP traffics at run-time. This could be valuable for system administrators to detect and identify the web intrusions as well.

To evaluate the framework and its prototype convincingly, we could conduct long term empirical research based on more web application experiments and more analysis technology comparisons. Evaluation could also be conducted on several real web applications which provide real services for customers. In this thesis, we only did the comparison between Drools Expert and Esper Engine. We could include more technologies that can be used to detect intrusions by implementing and plugging them into the tool prototype. We also could evaluation our framework based on performance testing, so that we are able to compare our technique with other existing approaches.

Furthermore, we could extend the usability of the IDPS framework to other areas of intrusion detection and prevention, not just for protecting web systems. For example, we could integrate the AB-IDPS with SONAR (Sustainable Optimization and Navigation with Aspects for system-wide Reconciliation) to detect and prevent the intrusions on lower levels, such as the operating system level and network level. SONAR is a dynamic diagnostic and optimization model combining Aspect-Oriented Programming, Extensible Markup Language, and management tools such as Java Management Extensions (JMX) [44]. SONAR can apply aspects to any level from the application level to the operating system level. Thanks to this, the AB-IDPS could detect and prevent intrusions at different levels based on SONAR's diagnostic data. The integration would make the AB-IDPS framework and its prototype more valuable for computing communities.

Bibliography

[1] D. Balzarotti, M. Cova, V. V. Felmetsger and G. Vigna, "Multi-module vulnerability analysis of web-based applications," In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS 2007),* pp. 25-35, 2007.

[2] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," In *Proceedings of the ACM Conference on Computer and Communication Security (CCS 2003),* pp. 251-161, 2003.

[3] D. Scott and R. Sharp, "Abstracting application-level web security," In *Proceedings of the International World Wide Web Conference (WWW 2002),* pp. 396-407, 2002.

[4] E. Kirda, C. Kruegel, G. Vigna and N. Jovanovic, "Noxes: A client-side solution for mitigating cross site scripting attacks," In *Proceedings of the ACM Symposium on Applied Computing*, pp. 330-337, 2006.

[5] G. Hermosillo, R. Gomez, L. Seinturier and L. Duchien, "AProSec: An aspect for programming secure web applications," In *Proceedings of the The Second International Conference on Availability, Reliability and Security,* pp. 1026-1033, 2007.

[6] N. Jovanovic, C. Kruegel and E. Kirda, "Pixy: A static analysis tool for detecting web application vulnerabilities," In *Proceedings of the 2006 IEEE Symposium on Security and Privacy,* pp. 258-263, 2006.

[7] M. Almgren, H. Debar and M. Dacier, "A lightweight tool for detecting web server attacks," In *Proceedings of the Network and Distributed System Security Symposium,* pp. 157-170, 2000.

[8] N. Jovanovic, E. Kirda and C. Kruegel, "Preventing cross site request forgery attacks," In *Proceedings of the IEEE International Conference on Security and Privacy for Emerging Areas in Communication Networks,* pp. 1-10, 2006.

[9] M. G. Uddin, H. Shahriar and M. Zulkernine, "ACIR: An aspect-connector for intrusion response," In *Proceedings of the 31st Annual International Computer Software and Applications Conference,* pp. 249-254, 2007.

[10] V. Haldar, D. Chandra and M. Franz, "Dynamic taint propagation for java," In *Proceedings of the Annual Computer Security Applications Conference,* pp. 303-311, 2005.

[11] G. Vigna, W. Robertson, V. Kher and R. A. Kemmerer, "A stateful intrusion detection system for World-wide web servers," In *Proceedings of the 19th Annual Computer Security Applications Conference,* pp. 34, 2003.

[12] W. Halfond and A. Orso, "AMNESIA: Analysis and monitoring for NEutralizing SQL-injection attacks," In *Proceedings of the International Conference on Automated Software Engineering,* pp. 174-183, 2005.

[13] AspectJ. in http://www.eclipse.org/aspectj/ (Accessed in December 2008).

[14] Drools. in http://labs.jboss.com/drools.html (Accessed in February 2009).

[15] EsperTech. in http://esper.codehaus.org/ (Accessed in February 2009).

[16] Hibernate. in https://www.hibernate.org/ (Accessed in February 2009).

[17] OWASP. "OWASP Top 10 2007," http://www.owasp.org/index.php/Top_10_2007, (Accessed in April 2008).

[18] D. Klein, "Defending against the wily surfer:Web-based attacks and defenses," in: *Proceedings of the USENIX Workshop on Intrusion Detection and Network Monitoring,* Santa Clara, CA, April 1999.

[19] K. Scarfone and P. Mell, "Guide to Intrusion Detection and Prevention Systems (IDPS)," NIST Special Publication 800-94, 2007.

[20] WebCohort Inc., "Only 10% of Web Applications are Secured Against Common Hacking Techniques," in http://www.imperva.com/news/press/2004-feb-02.html.

[21] Dagstuhl Seminar 08031 on Software Engineering for Self-Adaptive Systems, January 13-18, 2008, http://www.dagstuh1.de/08031.

[22] P. Robertson, "Self-Adaptive Software," in http://people.csail.mit.edu/paulr/ (Accessed in February 2009).

[23] H. A. Müller, M. Shaw, and M. Pezze. "Visibility of Control in Adaptive System," *Second International Workshop on Ultra-Large-Scale Software-Intensive Systems (ULSSIS 2008), Workshop at 30th ACM/IEEE International Conference on Software Engineering (ICSE 2008),* Leipzig, Germany, May 2008. In press.

[24] J. O. Kephart and D. M. Chess. "The Vision of Autonomic Computing," IEEE Computer, vol. 36, No. 1, pp. 41-50, January 2003.

[25] J. Kramer and J. Magee, "Self-Managerd system: An Architectual Challenge," FOSE 2007: 2007 Future of Software Engineering, 29th ACM/IEEE International Conference on Software Engineering (ICSE 2007), Minneapolis, Minnesota, USA, pp. 259-268, May 2007.

[26] D. Johnston-Watt. "Under New Management," ACM Queue, Vol. 4, no. 2, march 2006.

[27] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, J. Irwin. "Aspect-Oriented Programming," *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97)*. LNCS 1241. pp 220-242. June 1997. Springer-Verlag.

[28] J. Viega, J.T. Bloch and P. Chandri, "Applying Aspect-Oriented Programming to Security," Cutter IT Journal, Volume 14, No. 2, pp. 31-39, 2001 10.

[29] E. Armstrong, J. Ball, S. Bodoff, D. Carson, I. Evans, D. Green, K. Hasse, E. Jendrock, "The J2EE 1.4 Tutorial," http://java.sun.com/j2ee/1.4/docs/tutorial/doc/, December 2005.

[30] S. Hanenberg, R. Unland "A Proposal For Classifying Tangled Code," In: Workshop Aspekt-Orientierung der GI-Fachgruppe 2.1.9. Bonn, Germany 2002.

[31] P. Lin, A. MacArthur, J. Leaney, "Defining Autonomic Computing: A Software Engineering Perspective," Software Engineering Conference, 2005.

[32] IBM Corporation. "An Architectural Blueprint for Autonomic Computing," White Paper, 4th Edition, June 2006, http://www.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf.

[33] R. Laddad, "AspectJ in Action, Practical Aspect-Oriented Programming," Manning Publication Co., ISBN 1-930110-93-6, 2003.

[34] Java Caching System in http://jakarta.apache.org/jcs/

[35] E. Friedman-Hill, "Jess in Action, Rule Based System in Java," Manning Publication Co., ISBN 1-930110-89-8, 2003.

[36] David Luckham, "The Power of Events," Addison Wesley, ISBN 0-201-72789-7, 2005.

[37] Drools Expert in http://www.jboss.org/drools/drools-expert.html (Accessed in December 2009).

[38] Esper, http://esper.codehaus.org (Accessed in December 2009).

[39] Tomcat, http://tomcat.apache.org/download-55.cgi (Accessed in December 2009).

[40] Q. Zhu, "Goal Trees and Fault Trees for Root Cause Analysis," in *Proceedings IEEE International Conference on Software Maintenance (ICSM 2008)*, pp. 436-439, Beijing, China, 2008.

[41] PostgreSQL, http://www.postgresql.org (Accessed in December 2009).

[42] MySQL, http://www.mysql.com (Accessed in December 2009).

[43] Q, Zhu, "Adaptive Root Cause Analysis and Diagnosis using Complex Event Processing and Feedback Loops," Ph.D. Dissertation, to appear University of Victoria, 2010.

[44] C. R. Liu, C. Gibbs and Y. Coady, "Safe and Sound Evolution with SONAR," in *Transactions on Aspect-Oriented Software Development IV*, pp. 163-190, Springer-Verlag Berlin Heidelberg, 2007.

Appendix A: Glossary

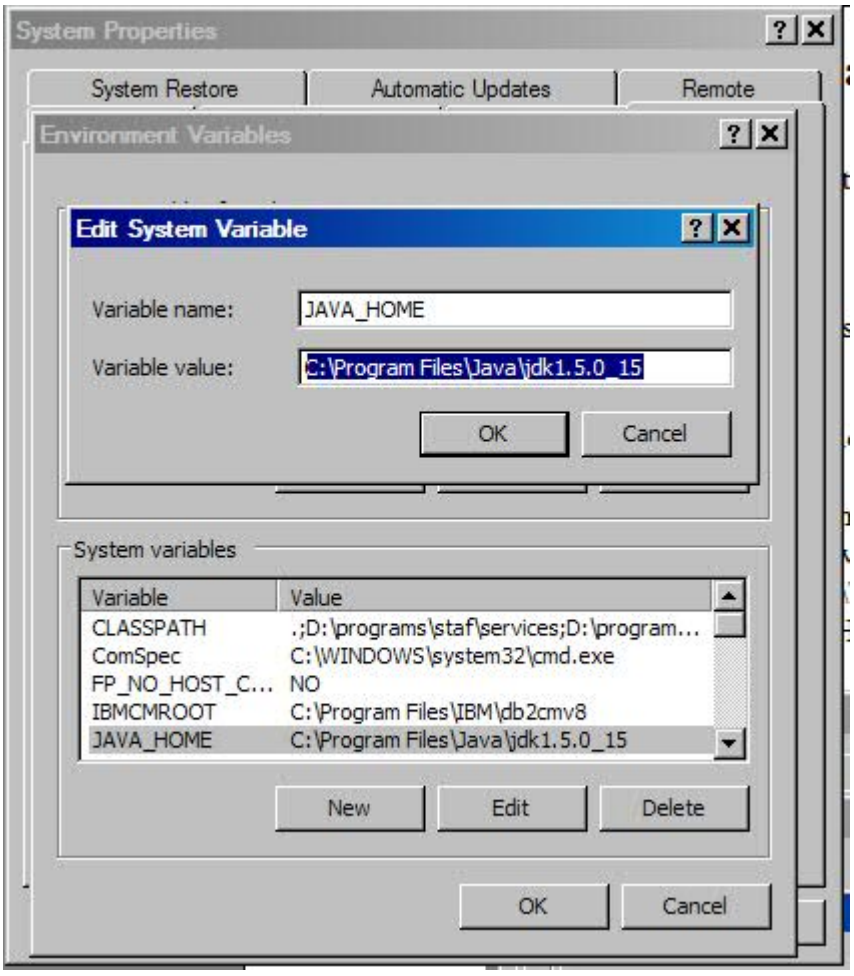
AB-IDPS	Aspect-Based Intrusion Detection and Prevent System.
AC	Autonomic Computing.
Advice	Additional code that can be applied to an existing program model.
AI	Artificial Intelligence.
Aspect	Combination of the pointcut and the advice.
AspectJ	An aspect-oriented extension created for the Java programming language.
API	Application Programming Interface.
AOP	Aspect-Oriented Programming.
BRMS	Business Rule Management System.
CEP	Complex Event Processing.
CSS	Cascading Style Sheet.
DayTrading	A web-based stock trading system that handles the customer stock operations such as buying and selling stocks on-line
Drools	A business rule management system (BRMS) with a forward chaining inference based rules engine.
EPN	Event Processing Network.
Esper	Esper is a component for CEP applications.
HTML	Hyper Text MarkUp language.
HTTP	Hypertext Transfer Protocol.
IDPS	Intrusion Detection and Prevention System.
IDS	Intrusion Detection System.
IPS	Intrusion Prevention System.
JBoss AS	A free software/open-source Java EE-based application server.
JCS	Apache Java Cache System.
JSP	Java Server Pages.
MySQL	A relational database management system that runs as a server providing multi-user access to a number of databases.
Online Photo Store	A web application that users can purchase photos and check their purchase history on-line.
OOP	Object-Oriented Programming.
OWASP	Open Web Application Security Project.
Pointcut	The point of execution in the application at which cross-cutting concern needs to be applied.

PostgreSQL	An free and open source object-relational database management system.
RCAD	Root Cause Analysis and Diagnosis.
Servlet	A Java class which conforms to Java Servlet API and is used to respond to HTTP requests.
SONAR	Sustainable Optimization and Navigation with Aspects for system-wide Reconciliation.
SQL	Structured Query Language.
Tomcat	An open source servlet container developed by the Apache Software Foundation.
UML	Unified Modeling Language.
XML	Extensible Markup Language.
XSS	Cross Site Scripting.

Appendix B: Installation and Set up Instructions

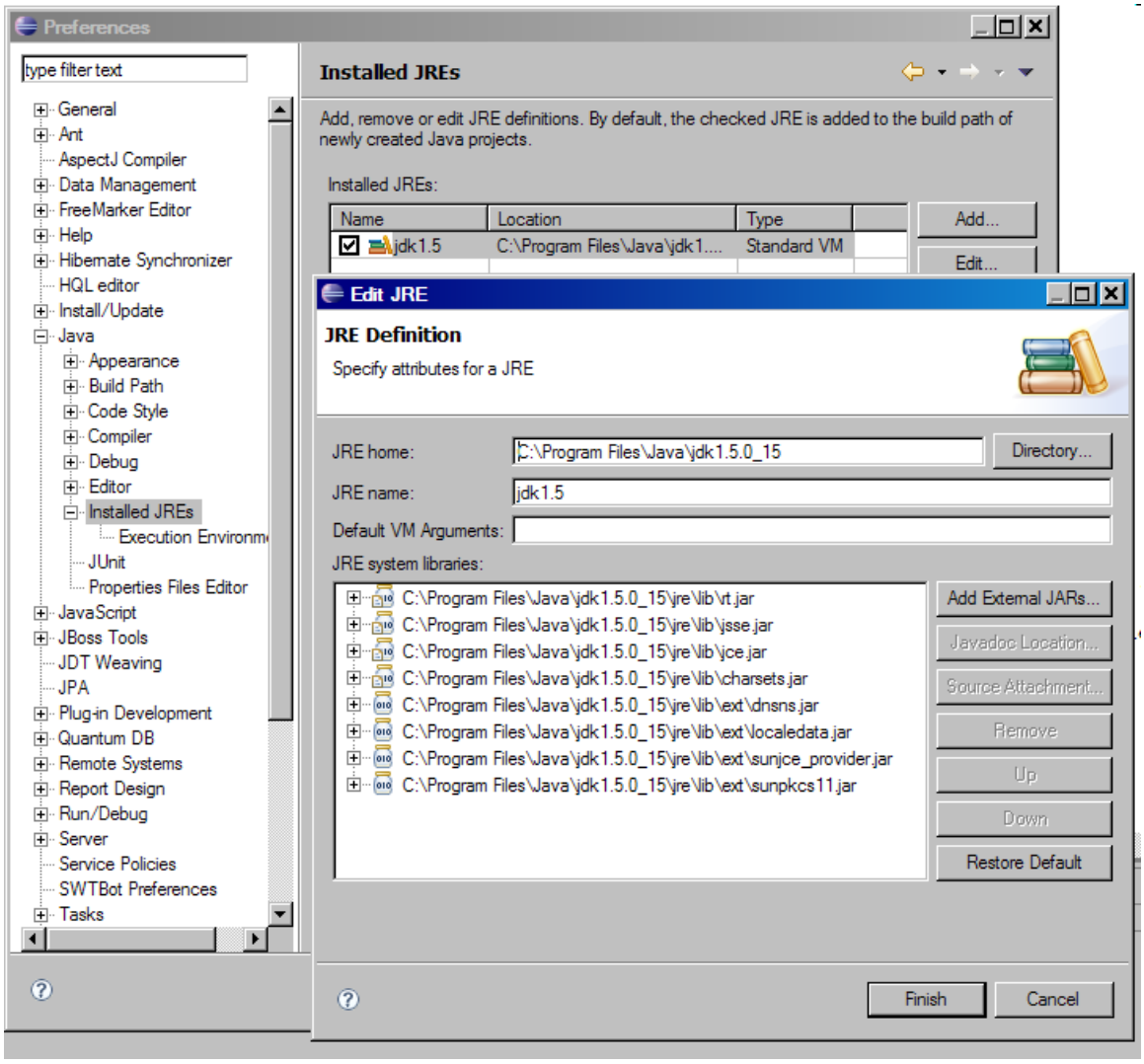
To experiment with the AB-IDPS prototype, the development and test environment has to be setup correctly. Please follow the step described below to set up the environment.

1) Copy all files from the installation directory
2) Create a folder JDK1.5
3) Create a folder JRE1.5
4) Install jdk-1_5_0_15-windows-i586-p.exe in the above directory
   - set up class path
   - right click on my computers in windows explorer
   - inside Environment variables
   - JAVA_ HOME = c:\\java\jdk1.5.0_15
   - PATH = ;%JAVA_HOME%;%JAVA_HOME%\bin



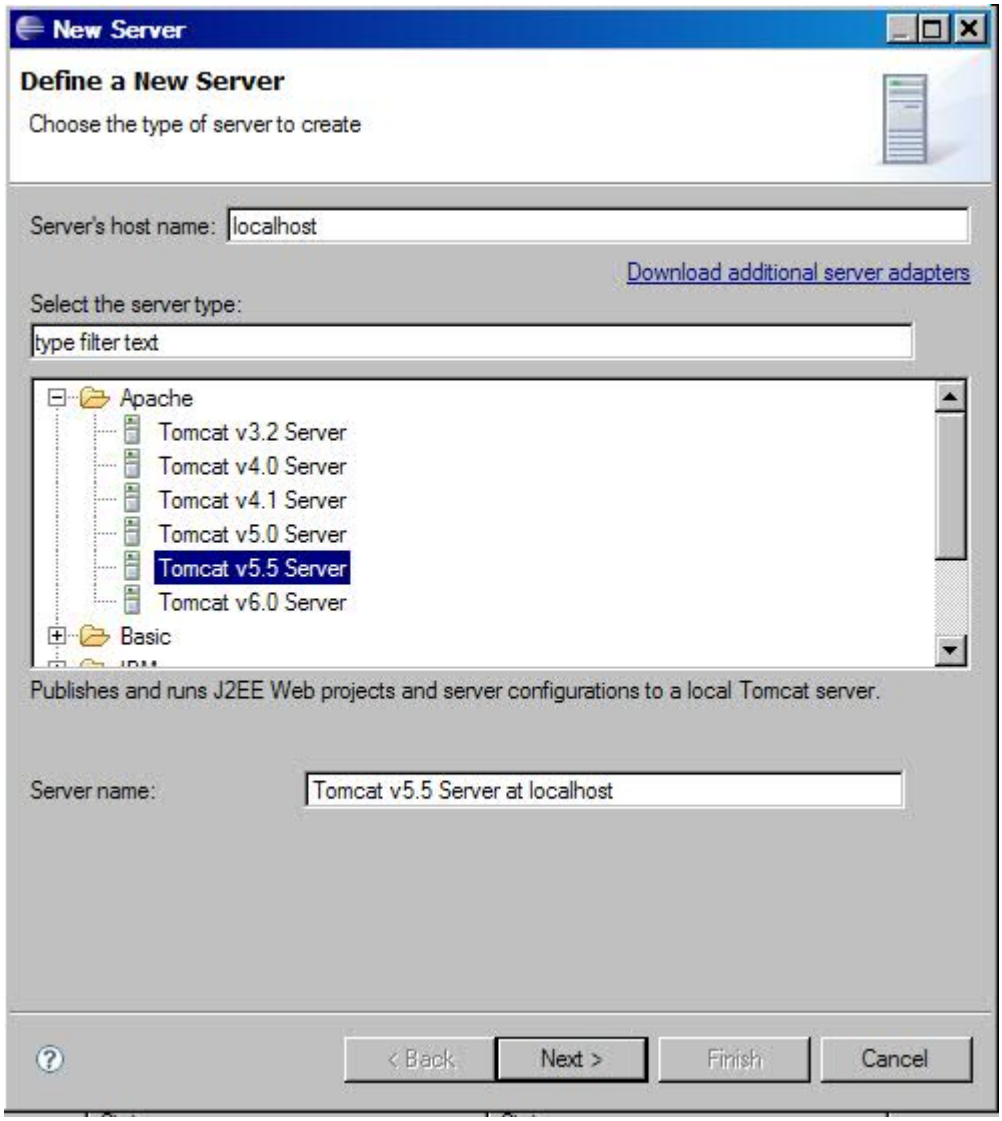
5) Restart computer
6) Create a folder WTP
7) Install wtp-all-in-one-sdk-R-1.5.2-200610261841-win32.exe

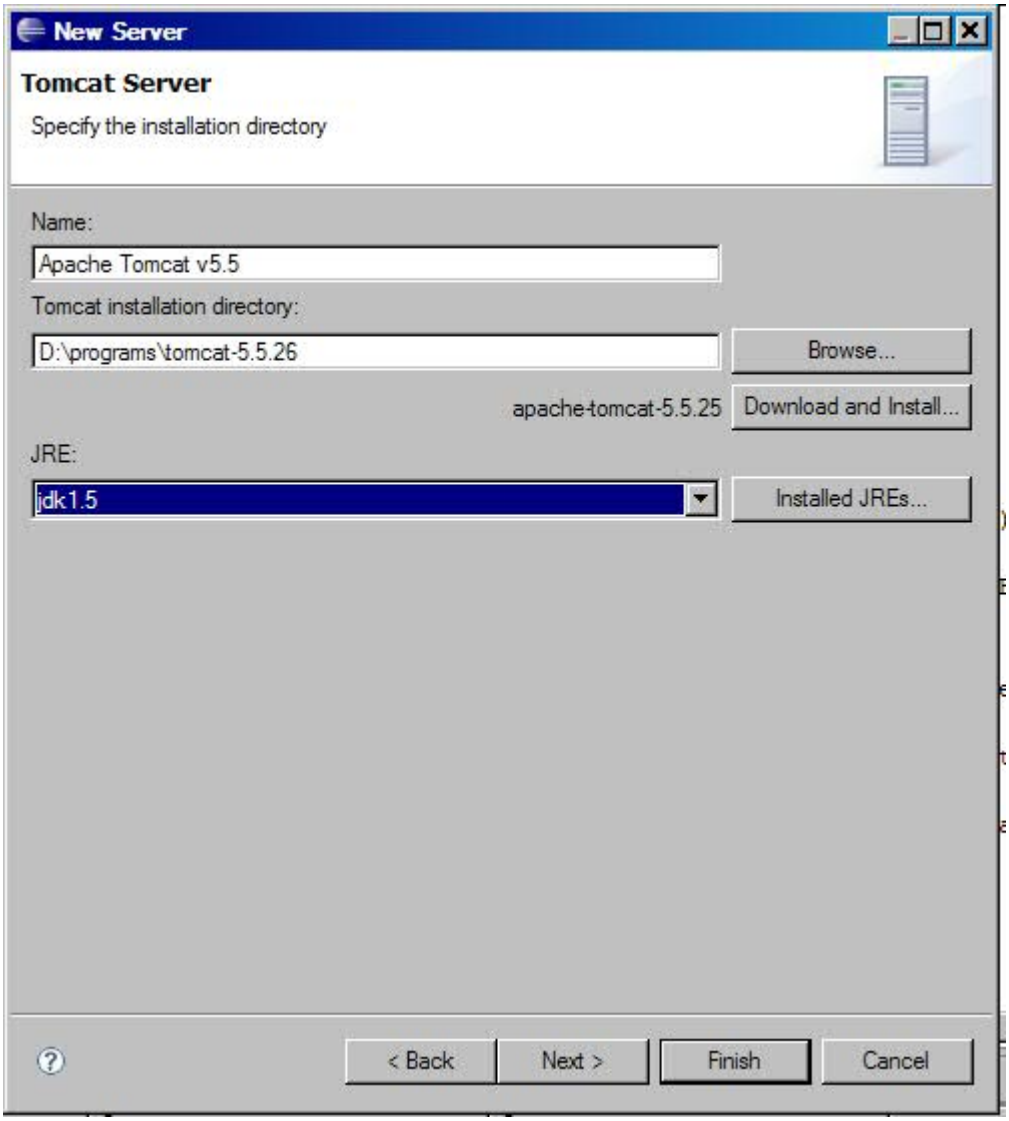
- Install server
- Eclipse window --> Preferences -->Server-->Installed Runtimes
- check Installed JRE



8) Install Apache Tomcat 5.5.26

- Unzip tomcat jakarta-tomcat-5.5.26.zip in D:\programs\tomcat-5.5.26
- Eclipse Window--> Show view-->Other--> Server-->Servers
  Servers console--> New Server-->Tomcat v5.5 Server--> Finish
  Test Server by starting it.

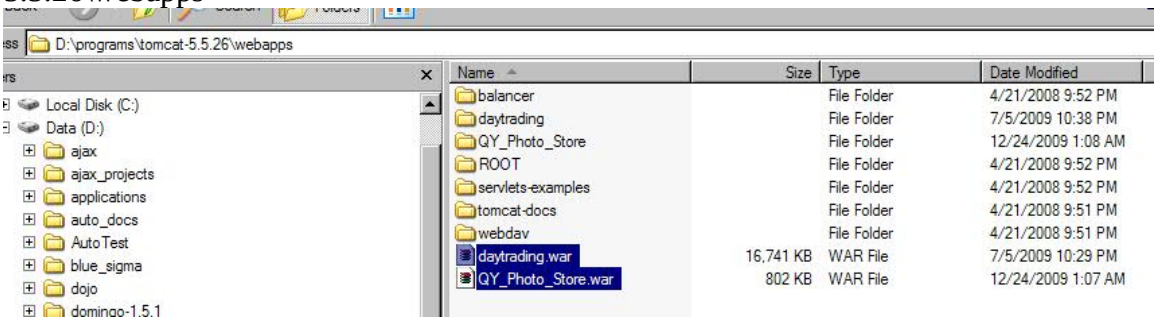

New Server
Define a New Server
Choose the type of server to create
Server's host name: localhost
Download additional server adapters
Select the server type:
type filter text
Apache
Tomcat v3.2 Server
Tomcat v4.0 Server
Tomcat v4.1 Server
Tomcat v5.0 Server
Tomcat v5.5 Server
Tomcat v6.0 Server
Basic
Publishes and runs J2EE Web projects and server configurations to a local Tomcat server.
Server name: Tomcat v5.5 Server at localhost
< Back
Next >
Finish
Cancel



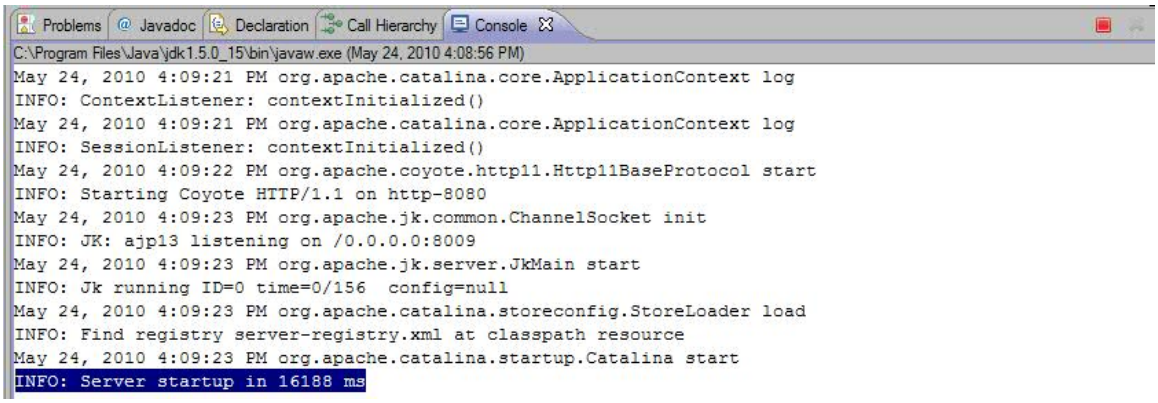
9) Install PostgreSQL 8.2

- Install server
- Create two databases: daytradingDB and idpsDB.

10) Copy daytrading.war and QY_Photo_Store.war to D:\programs\tomcat-5.5.26\webapps



11) Copy ABIDPS.jar to D:\programs\tomcat-5.5.26\common\lib
12) Copy ABIDPS.properties to D:\programs\tomcat-5.5.26\conf
13) Start Tomcat from Eclipse



14) Open Internet Explorer
15) Connect to http://localhost:8080/daytrading

16) Start the experiments using the XSS or SQL Injection attack strings.