

A SERVER-SIDE SECURITY MODEL FOR
WEB APPLICATIONS

BY

JING DING

Bachelor of Science

Hefei United University

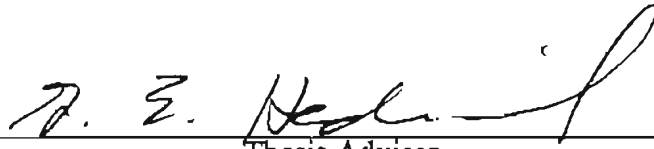
Hefei, P.R.China

1995

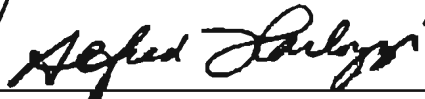
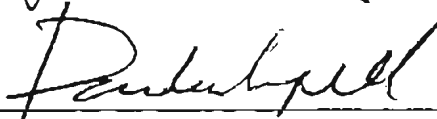
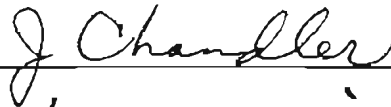
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in Partial Fulfillment of
The Requirements for
The Degree of
MASTER OF SCIENCE
December 2003

A SERVER-SIDE SECURITY MODEL FOR
WEB APPLICATIONS

Thesis Approved:



Thesis Adviser



Dean of the Graduate College

ACKNOWLEDGEMENTS

I would like to acknowledge the continued support and guidance from my thesis adviser, Dr. G. E. Hedrick. He provided deep insight and technical advice on all issues of this research.

I would also take this opportunity to convey my sincere thanks to Dr. Huizhu Lu for her suggestions during the preparation of my research. I am grateful to Dr. J. P. Chandler, Dr. N. Park and Dr. M. H. Samadzadeh for taking out their time and for providing constructive feedback for my research as my committee members.

I wish to express my appreciation to the Computer Science Department at Oklahoma State University for supporting me during my studies.

Finally, I would like to thank my husband Yong Hu, for his love, patience, and endurance that made this journey a pleasant experience.

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION	1
1.1 Motivation	1
1.2 Objectives.....	2
1.3 Organization of the thesis.....	3
II. LITERATURE REVIEW.....	4
2.1 The Concept of Web Applications.....	4
2.2 Issues of Web Application Security.....	5
2.3 Methods for Authentication	5
2.4 Methods for Authorization.....	6
2.5 Session Tracking.....	7
2.6 Firewall.....	11
2.7 Intrusion Detection System.....	11
2.8 Secure Socket Layer.....	12
2.9 Sanitizing Browser Inputs.....	13
2.10 Sensitive Web Page Caching Prevention.....	13
2.11 Cryptography.....	14
III. A SERVER-SIDE SECURITY MODEL.....	16
3.1 Input Web Page Domain Checking.....	16
3.1.1 Definition	16
3.1.2 Implementation.....	17
3.2 Cookie Support Detector.....	21
3.3 Proposed Server-Side Security Model	22
3.3.1 Problems of Existing Security Approaches.....	22
3.3.2 Proposed Server-Side Security Model	25
IV. WHEAT BIN MIX OPTIMIZATION WEB APPLICATION.....	30
4.1 Objective	30
4.2 Background Information	31

Chapter	Page
4.2.1 Terminologies.....	31
4.2.2 Standard Wheat Grade Table	33
4.2.3 Discount Table	34
4.2.4 Sketch of Web Bin Mix Optimization Web Application.....	37
4.2.5 A Limitation of The Optimization Algorithm.....	38
4.2.6 Expansion Capability	39
4.3 User Interface	39
4.4 Inside of Wheat Bin Mix Optimization Web Application	49
4.4.1 Architecture.....	48
4.4.2 Tools Used	51
4.4.3 Pattern Search Optimization Algorithm.....	52
V. SECURITY MODEL IN WHEAT BIN MIX OPTIMIZATION WEB APPLICATION.....	53
5.1 Architecture.....	53
5.2 New Service Components.....	55
5.3 Server-Side Security Model Implementation.....	67
5.3.1 Defining Input Web Page Domain.....	67
5.3.2 Implementation.....	68
VI. SUMMARY AND FUTURE WORKS	77
6.1 Summary	77
6.2 Future Works.....	78
BIBLIOGRAPHY	79
APPENDIX: JSP CODES AND JAVA PROGRAMS FOR SECURITY LAYERS.....	82

LIST OF FIGURES

Figure	Page
2.1 An Example of Role Hierarchy And Associated Privilege	7
3.1 “Hello Shop” Web Application Workflow	18
3.2 Cookies Support Detector	22
3.3 A Server-Side Security Model For Web Applications.....	26
4.1 Home Web Page.....	40
4.2 Bin Information Web Page.....	41
4.3 List Bin Information Web Page	42
4.4 List Bin Information From A File Web Page.....	43
4.5 Optimization Setting Pop-Up Window	44
4.6 Optimization Setting Error Pop-Up Window.....	45
4.7 Optimization Result Pop-Up Window	46
4.8 Rank Criteria Setting Pop-Up Window.....	47
4.9 Rank Pop-Up Window	47
4.10 Grade Table Pop-Up Window.....	48
4.11 Discount Table Pop-Up Window	48
4.12 The Architecture of Wheat Bin Mix Optimization Web Application.....	50
5.1 The Architecture of Protected Wheat Bin Mix Optimization Web Application	54
5.2 BinWebSite Web Page	56

Figure	Page
5.3 SetCookies Web Page	57
5.4 TestCookies Web Page.....	58
5.5 Error2 Web Page	59
5.6 Error1 Web Page	59
5.7 Login Web Page	60
5.8 Check Web Page	61
5.9 LoginError Web Page.....	62
5.10 NoChance Web Page.....	62
5.11 CustRegist Web Page	63
5.12 RegistDisplay Web Page.....	64
5.13 MustGive Web Page.....	64
5.14 RedoRegist Web Page.....	65
5.15 EditDisTable Pop-up Window	66
5.16 EditGradeTable Pop-up Window.....	67
5.17 Digital Signature Generation.....	72
5.18 Digital Signature Verification	73

CHAPTER 1

INTRODUCTION

1.1 Motivation

Web applications are widely used to access database systems for information retrieval, transactions and publication. They are commonly applied for e-commerce, e-bank, e-education, and e-government etc. We can use web applications to purchase goods, to transfer funds, to enroll courses, to retrieve academic transcripts, and to pay taxes, etc. Most of those web applications process, store, and transmit sensitive data. Therefore, protecting sensitive data becomes the most important thing for those web applications' design.

Authentication and Authorization are the two key security issues when considering web application security. Authentication is used to authenticate a person is really the one he said he is. After authentication, authorization is used to ensure that each authenticated user only accesses information he is allowed to access.

Today's authentication standard method is Username/Password authentication. In authorization, Role Based Access Control is the approach most widely used to prevent secure information from unauthorized access. However, what can we do if hackers bypass authentication? What can we prevent hackers from doing 'Forceful Site Browsing' (access sensitive information through a direct URL)? Username/Password authentication is similar to security guards watching the lobby of a building. They restrict access to the building if someone want to come into the building through the door, but

they do nothing to control a person who tries to get into the building through a tunnel. Then what happens once the person successfully gets into the building through a tunnel? Access control is used to ensure that every authenticated user only to access information he or she has the privilege to access. For those attackers who bypass authentication successfully, access control exists in name only.

Obviously, it is urgent for us to find some other security techniques to protect web applications in addition to Username Password authentication and Role Based Access Control.

1.2 Objectives

There are three objectives of the thesis:

- Compare the pros and cons of existing security approaches used to secure web applications.
- Propose a server-side security module for web applications to meet the following security requirements:
 - Prevent malicious users from accessing sensitive information without authentication.
 - Prevent authenticated users from accessing information beyond their authorization.
- Construct a web application to demonstrate the proposed security model.

1.3 Organization of the thesis

This thesis comprises the following chapters: Chapter 2 is the literature review. In Chapter 3, a server-side security model for web applications is proposed. Two security techniques, Input Web Page Domain Checking and Cookie Support Detector, are also proposed. The proposed server side security model consists of eleven security layers, where the two proposed techniques act as two security layers. Chapter 4 describes in detail the construction of wheat bin mix optimization web application. This web application is used to demonstrate the proposed server-side security model. Chapter 5 addresses how to build the proposed server-side security model into wheat bin mix optimization web application. Chapter 6 is the summary and future works.

CHAPTER 2

LITERATURE REVIEW

In this Chapter, we present some background information such as the concept of web applications, security issues and security measures currently used to protect web applications.

There are eleven sections in this Chapter. Section 2.1 introduces the concept of web applications. Section 2.2 discusses security issues of web applications. Section 2.3 describes some methods to implement authentication. Section 2.4 shows the most widely used authorization method—Role Based Access Control. Session tracking problems are discussed in section 2.5. Some other security techniques such as Firewall, Intrusion Detection System, Secure Socket Layer, Sanitizing Browser Inputs, and Sensitive Web Page Caching Prevention are introduced in section 2.6, 2.7, 2.8, 2.9, 2.10, respectively. The last section 2.11 describes some encryption and decryption methods used in web application security.

2.1 The Concept of Web Applications

“Web applications are the business logic that enables users’ interaction with the web site, and the transacting and interfacing with all the back-end data systems [Pettit 2001].” For example, applications allow users to check their account balance and to transfer funds; applications that allow users to shop online; applications that allow students to enroll classes, and many, many others.

Typically, a web-based application can be represented in a three-tier architecture, which includes a web-client, network servers, and a back-end information system supported by several databases.

2.2 Issues of Web Application Security

There are two key issues in the web application security: *Authentication* and *Authorization*. *Authentication* is used to ensure that someone is exactly the person he said he is. It is the process of allowing only valid (authenticated) web visitors to view web pages of a web application. *Authorization* is used to ensure that authenticated user can only do things what he is authorized to do. There are many kinds of methods to implement the two key issues.

2.3 Methods for Authentication

- Password-based authentication [George 1997]
- Host-based authentication [George 1997] [Cook 2000] [Berry 1994]
- Public Key Infrastructure (PKI) based authentication [Oracle Company 1999]
- Other third party-based authentications:
 - Kerberos [Neuman 1994]
 - Distributed Computing Environment (DCE) [Rosenberry 1992]
 - Smart Card [Rankl 1997]

Among all of these methods, password-based authentication is today's authentication standard and is used most widely since it is easy for human beings to grasp

and no additional hardware is needed. Usually, password-based authentication asks a user to give the correct user name and the corresponding password.

2.4 Methods for Authorization

Joshi, Aref, Ghafoor, and Spafford surveyed all access control models used nowadays with their key features and approaches [Joshi 2001]. They also concluded "The Role-Based Access Control (RBAC) model is expected to provide a viable framework for addressing a wide range of security requirements for large enterprise [Joshi 2001]."

Role Based Access Control [Ferraiolo 2003] is today's most widely used authorization method. In RBAC model, each user is assigned one or more roles; each role is assigned one or more privileges; roles can be organized in hierarchies.

For example, in Figure 2.1, we can divide employees of a university into two groups: administration staff and research staff. The administration staff consists of secretary, college dean and department Chair. The research staff consists of researcher, faculty and department Chair. Notice that department Chair belongs to both administration staff group and research staff group.

Now suppose that all employees are authorized with privilege A, administration staff is authorized with privilege B, and research staff is authorized with privilege C. Then a secretary has privilege A and B, so does the college dean. Researchers and faculty have privilege A and C. Since the department Chair has both the administration staff and research staff role, the department Chair has privilege A, B and C.

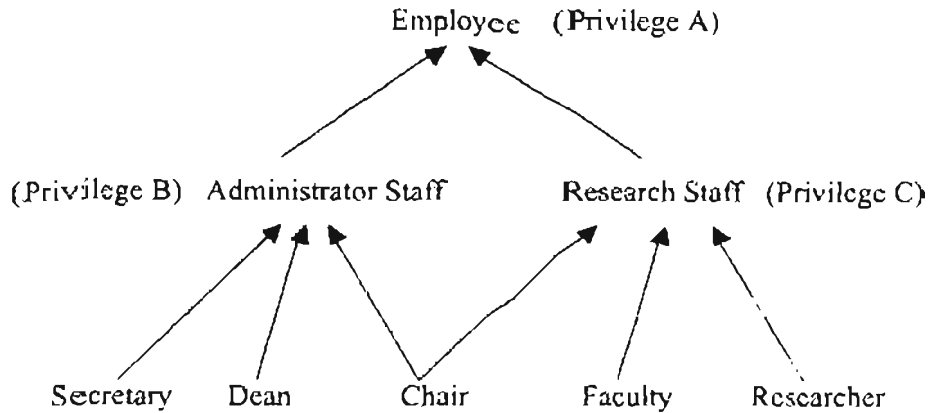


Figure 2.1 An Example of Role Hierarchy And Associated Privilege

2.5 Session Tracking

The client-server model basically follows three procedures: requests from the client side, the server side's responses and afterwards, the client side's acknowledgement.

This is perfect for simple web browsing, where each request typically results in a web page being sent back to the client. The server does not need to know whether a series of requests come from the same, or from different clients, or whether those requests are related or distinct. However, when writing web applications, these are things that we may concern.

The idea of maintaining state among requests to a web application is known as *session tracking*. A *session* can be defined as a series of related interactions between a single client and the web server that take place over a period of time. No matter what authentication and authorization methods are chosen, there are also strong needs to ensure that a session related with each user is secure so that authentication and authorization methods can be brought into full play.

In the following sections, several traditional session tracking methods are discussed.

URL Rewriting

URL is the abbreviation of Universal Resource Locator. URL Rewriting basically means that when a user is presented with a link to a particular resource instead of simply presenting the URL as normally do, the URL for that resource is modified so that more information is passed when requesting for that resource [Ayers 1999].

Assume that a user searches some books from an on-line bookstore www.bookstore.com and he is presented with a search result that has 2 books listed. Suppose the search result displayed to the user is basically a form within Java Server Page (JSP) format:

```
<form method = "post" action = "book.jsp">  
<input type = "checkbox" name = "bookID" value = "1"> C++<br>  
<input type = "checkbox" name = "bookID" value = "2"> JAVA<br>  
<input type = "submit" name = "Submit" value = "Add To Cart"><br>  
</form>
```

In this form, there are two checkboxes, each for one book and a Submit button. The user can click any checkbox to add any of these books to his Cart.

In this example, when a user links to the on-line bookstore www.bookstore.com, the URL appears in the user's web browser is <http://www.bookstore.com/book.jsp>, where the server is www.bookstore.com, the server resource is `book.jsp` (a Java Server Page file). After the user clicks the checkbox of the book "C++", the URL that appears in the

user's web browser is `http://www.bookstore.com/book.jsp?bookID=1`. The server source has been changed from `book.jsp` to `book.jsp?bookID=1`. This is exactly what URL Rewriting means. The effect of this is that any part of the URL after the "?" (question mark) is treated as extra parameters that are passed to the server side program.

URL Rewriting is very easily to maintain session information when a user's browser doesn't support cookies or a user disables cookies. We will introduce cookies technique later.

Hidden Form Field

Hidden Form Field is a session tracking technique in which browser input is not displayed when read by a web browser [Duffey 2001]. In this technique, session data can be tracked by storing it in hidden form fields then be retrieved later.

Cookies

Authentication is the first security layer of web application protection. After authentication, the server side must ensure that a user is still the same person who has successfully logged in. If a web application frequently asks users to do authentication, sometimes as often as every page request, this indeed makes the web application secure. However, it is unacceptable since it is not convenient for users.

Today's most widely used cookie technique makes authentication more convenient to users. HTTP by itself is stateless, after authentication, cookies can be the way a web application maintains session state information. Cookies are sets of strings written to a client's web browser or stored on a client's hard disk by the web application

server whenever that web browser visits the server's site. As a user visits a web site for the first time, the server side creates a new session and sets cookies with a unique value. When the user browses that web site again the cookies is sent back from the user's web browser to the server side, allowing the server side to recognize the user. Thus, cookies commonly are used to maintain state information among subsequent HTTP requests and can be exchanged between the web client and the web server to maintain connection information. Cookies usually are used to store information such as a user's host name, password, account ID, session ID or other profile information.

There are two types cookies:

- *Persistent cookies*: have an expiration date and are stored on a user's hard disk until that date. A persistent cookie can be used to track a user's browsing habits by identifying the user whenever the user returns to a site.
- *Non-persistent cookies (temporary cookies)*: are stored in the web browser's memory. They last only until the browser is closed or a user's session is over then are destroyed

Role-Based Access Control with Secure Cookies

Cookies are insecure to store and transmit sensitive information. Park, Sandhu and Ahn proposed Role-Based Access Control with persistent secure cookies technique to protect web applications [Park 2001]. In this technique, they suggested to use some cryptographic technologies, such as "digital signature" which we will introduce in section 2.11, to encrypt cookies to prevent cookies from theft or modification.

Session Timeout

Session timeout specifies the “no activity” duration beyond which a user has to re-authenticate himself to a web site. The “no activity” duration setting is usually based on the type of a web application. Serious financial web applications may specify a very short session timeout period. Regular applications, such as e-mail, may use longer timeout periods.

2.6 Firewall

Web servers are the places where most network services are located. Firewall technology has become the most popular defense for these servers against the open untrusted Internet.

“A firewall is a form of access-control technology that prevents unauthorized access to information resources by placing a barrier between an organization’s network and the Internet [Xtream 2002].” Deploying firewalls is a standard step adopted by many organizations. Firewalls protect against many attacks on the network and system infrastructure. In addition, some firewalls provide filtering capability and prevent inbound malicious applications.

2.7 Intrusion Detection System (IDS)

Intrusion detection is a technology that attempts to discover attacks, preferably while they are still under way. There are two approaches for this technique: *pattern-based* and *anomaly-based*.

Pattern-based systems are explicitly programmed to detect certain known kinds of attack. Anomaly-based systems address those attack problems by attempting to detect any abnormal behavior [Stillerman 1999].

2.8 Secure Socket Layer (SSL)

After a user has supplied proper identification and access is granted, Secure Socket Layer ensures secure data transmission during a session so that that private data is not intercepted or altered during the session.

HTTP is the basic protocol for data transmission on the Internet. The protocol was not designed for security, thus very insecure. For example, students in a dorm share a broadcast Ethernet. A user may impersonate another user by running a listening program. Therefore, even though a user has properly logged onto a system, any information that is accessed can be intercepted and captured by another user on the network. There is no easy way to prevent this interception except by encrypting all of the information that flows both ways.

Secure Socket Layer (SSL) technology is a solution that uses public key technology (section 2.11) to ensure that information exchanged between the web server and the web client is encrypted. SSL provides data transmission security between the web client and the web server. Accessing web sites using SSL appears different from regular sites. A normal web site may be: `http://www.website.com/`. If the web site uses SSL, then the 'http' protocol token becomes 'https' as in: `https://www.website.com/` [Duffey 2001]."

2.9 Sanitizing Browser Inputs

Browser inputs are all inputs from a web browser such as users' input data from HTML forms and cookies retrieved from the client's side etc.

Some special characters in HTML form inputs such as !, &, -, and \$ can cause a web server to execute an operating system command or has other unexpected behavior. Some web guest books allow users to format their comments with HTML tags such as . However, hackers may embed malicious HTML tags in the client requests. This is also called *cross-site scripting*. With cross-site scripting, malicious users can access and delete files stored on a web server, crash a user's computer or steal information from HTML fields [Advosys Consulting 2002].

It is very important for the server side to sanitize browser inputs. The server side should be able to strip unwanted characters, invisible characters and HTML tags from users' inputs. The best solution is to check browser inputs against a list of server-side defined valid characters other than a list of invalid ones since it is very difficult to determine all possible malicious characters.

2.10 Sensitive Web Page Caching Prevention

Web browsers cache pages. They store a local copy of every page a user visits on the Web. Caching speeds up a user's access to Web pages.

For example, if a user asks the web browser to get a web page visited before, the web browser first looks in the cache. If the browser finds that web page in the cache, it just loads that page rather than connects to a web server to get a new one.

However, in some cases, we may not want sensitive web pages to be cached by web browsers, such as some personal information on registration web page. In this situation, we need to prevent sensitive web page from being cached by web browsers to protect sensitive information.

2.11 Cryptography

Secret-key Encryption (using a single key) [Schwartz 2001], also known as Symmetrical-key Encryption, is somewhat familiar to most people. In Secret-key Encryption, a single key is used for both encryption and decryption. Public-key Encryption [Naor 1990] is a little more complicated. In Public-key encryption, each individual holds two keys, one public key and one private key. The public key is freely published, and the private key is kept private. Once a message is encrypted with one key, it cannot be decoded without the other key.

MD5 message-digest algorithm [Loshin 1999] is a one-way encryption algorithm. The algorithm takes a message of arbitrary length as input. The output is a 128-bit “fingerprint” or “message digest” of the input.

Secure Hash Algorithm (SHA-1) [Loshin 1999] is a message digest algorithm. It is the Federal Information Processing Standard. Given an input message of any length less than 2^{64} , the SHA-1 produces a 160-bit output called Message Digest. Then the Message Digest can be used as an input to a signature algorithm that generates or verifies the signature of the message.

Digital Signature Algorithm (DSA) [Yen 1995] is for creating and verifying signatures. It can also be used to produce a key pair: private key and public key. The pair

of key's size range is 512 to 1024 bits. Before generating and verifying a digital signature, we first use DSA to generate a public key and a private key. To generate a digital signature, we use SHA-1 to generate a message digest for a given message. Then we use DSA to generate digital signature for the message digest, which takes the message digest and the private key as input parameters. To verify a signature, we first re-compute the message digest for the original message. Then we use DSA to verify the digital signature by taking the message digest, the digital signature and the public key as input parameters. If a message is not modified, the verification result is true. Otherwise, the verification result is false.

The Rivest, Shamir, and Adleman Asymmetric Cipher Algorithm (RSA) [Loshin 1999] is a public key cryptosystem in order to ensure that nobody, except the intended recipient, deciphers the message.

Now we have discussed various methods used in web application security and their advantages. In Chapter 3, we will also discuss the disadvantages of these existing security techniques. By taking the advantages of those existing security techniques and combining them, along with two proposed techniques (Input Web Page Domain Checking and Cookies Support Detector), we propose a server-side security model in Chapter 3.

CHAPTER 3

A SERVER-SIDE SECURITY MODEL

In this Chapter, we will introduce a proposed sever-side security model for web applications. This server-side security model consists of eleven security layers, where two layers are the two proposed security techniques, Input Web Page Domain Checking and Cookies Support Detector, which will be described in section 3.1 and section 3.2 respectively. Section 3.3 discusses the disadvantage of several existing security techniques and explains the purpose of each security layer in the proposed server-side security model.

3.1 Input Web Page Domain Checking

As we known, web applications are based on the client-server architecture. For each client request, the server sends corresponding web page from the server side to the client side. There are some particular orders when accessing web pages. We call this particular order as Application Workflow.

Input Web Page Domain Checking technique is proposed to ensure the correct application workflow. Therefore, this technique can prevent malicious user from accessing sensitive information without authentication such as forceful browsing.

3.1.1 Definition

Based on the particular order while accessing web pages, when requests to access a certain web page are received, the server side must ensure that only those requests coming from some particular web pages will be processed, and others will be declined. "Input Web Page Domain" is a set of such particular web page(s). Each web page owns its own "Input Web Page Domain".

3.1.2 Implementation

Assume there is a member-only web application called "Hello Shop" designed for an on-line store. Every customer uses this web application for shopping should have a membership in the on-line store.

Suppose there are totally three Web pages designed for "Hello Shop" Web application: A "Login" Web page is used to authenticate users; a "Product List" Web page displays all products; a "Shopping Cart" web page lists all items a customer chose. We can do "Input Web Page Domain Checking" following three steps:

Step1: Drawing Workflow during Web Application Design

For the "Hello Shop" Web application, the "Log In" web page should be sent to the client's side (a user) from the server side to authenticate users before any other Web pages. In the "Log In" web page, a user types his or her username and password to login. If the user's given information is incorrect, i.e., incorrect username and/or incorrect password, the "Log In" page should be resent to the client's side for re-login. Otherwise, the "Product List" web page should be sent from the server side to the client side. If a user logs in successfully, he or she becomes an authenticated user. The user now can

choose a product and add it to the user's shopping cart by clicking the "Add To Cart" button in the "Product List" web page. The "Shopping Cart" web page should then be sent from the server side to the client's side to display all items currently in the shopping cart. If the user wants to continue shopping, the "Product List" web page should be re-sent from the server side to the client's side. Based on the particular orders of the three web pages, we can draw the workflow of "Hello Shop" Web application, as shown Figure 3.1.

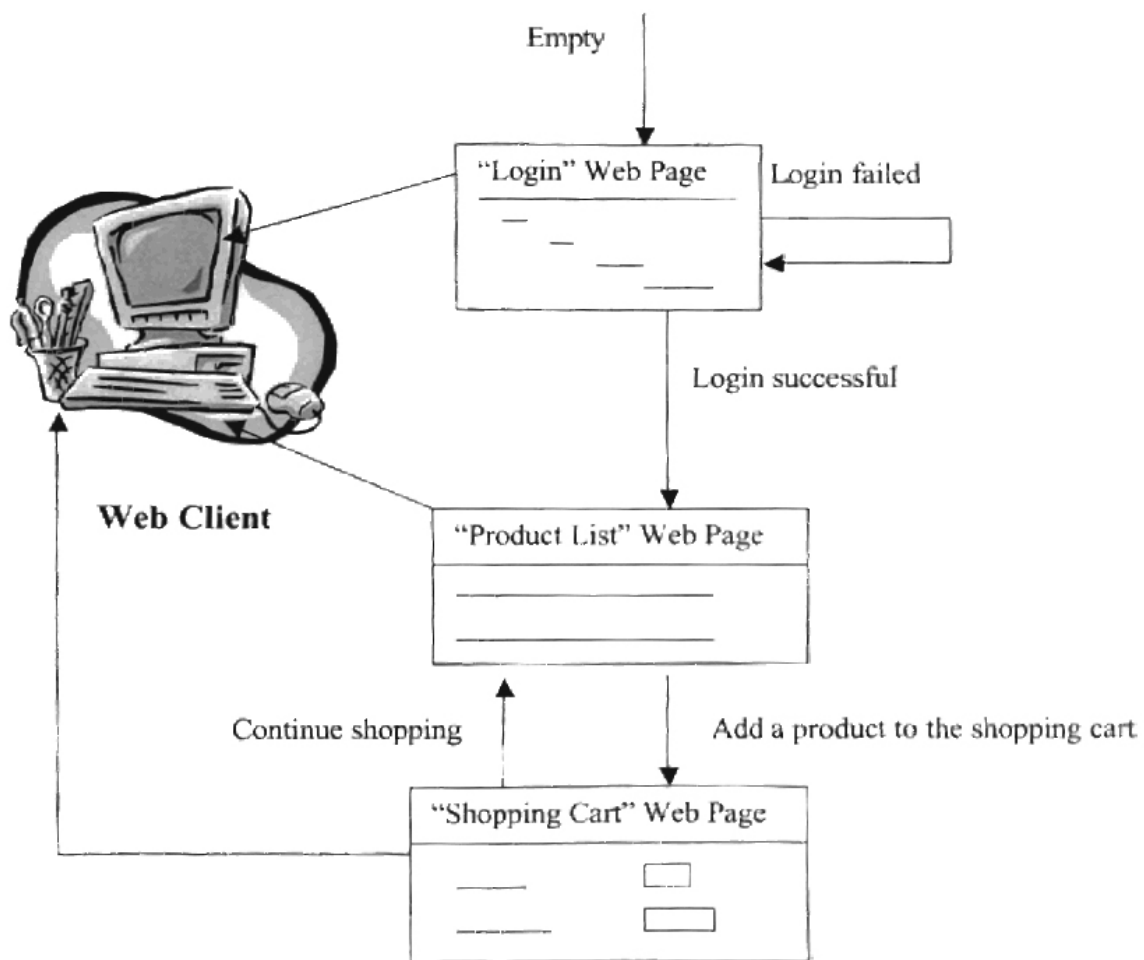


Figure 3.1 "Hello Shop" Web application workflow

Step 2: Define “Input Web Page Domain” for Each Web Page

Based on the workflow of “Hello Shop” web application we drew in step 1, we can define the “Input Web Page Domain” for each Web page as in Table 3.1.

Table 3.1 “Input Web Page Domain” for “Hello Shop” Web Application

Page Name	Input web page domain
Log In	{Null, “Log In”}
Product List	{“Log In”, “Shopping Cart”}
Shopping Cart	{“Product List”}

- Define “Input Web Page Domain” for “Log In” web Page:

“Log In” web page should be sent first from the server side to the client side before any other web pages for authenticating a user. We use “Null” as an element in the “Input Web Page Domain” of “Log In” web page to show that “Log In” web page must be the first web page sent to the client’s side.

If a user fails to log in, the “Log In” web page is resent to the client side. Thus, “Log In” web page itself is an element in the “Input Web Page Domain” of “Log In” web page.

Therefore, “Input Page Web Page Domain” for the “Log In” web page is {Null, “Log In”}.

- Define “Input Web Page Domain” for “Product List” Web Page:

“Product List” web page can be redirected by the “Log In” web page if a user logs in successfully; “Product List” web page can also be redirected by the “Shopping Cart”

web page if a user wants to continue shopping after adding a product into the shopping cart.

Therefore, the “Input Web Page Domain” for “Product List” web page contains only “Log In” web page and “Shopping Cart” web page. We denote this “Input Web Page Domain” as {“Log In”, “Shopping Cart”}.

- Define “Input Web Page Domain” for “Shopping Cart” Web Page:

Since “Shopping cart” web page can only be redirected from the “Product List” web page and can't be redirected by the “Log In” web page, the “Input Web Page Domain” for the “Shopping Cart” web page only contains the “Product List” web page. We use {“Product List”} to denote the “Input Web Page Domain” for the “Shopping Cart” web page.

Step 3: Do “Input Web Page Domain” Checking for Each Web Page

Do “Input Web Page Domain” checking can according the following rules:

We define:

- P: A web page to be requested
- D: P's input web page domain
- Q: A web page that makes the request

Then the checking is performed according to the following rule:

If $Q \in D$, process the request. Otherwise, decline the request.

In the “Hello Shop” web application, since the “Input Web Page Domain” for “Log In” web page is {Null, “Log In”}, we must check to ensure that no other web page is sent to the client side before the “Log In” web page from the server side except possibly the “Log In” web page itself.

Since the “Input Web Page Domain” for “Product List” web page is {“Log In”, “Shopping Cart”}, only requests coming from the “Log In” web page and requests coming from the “Shopping Cart” web page can be processed. If requests come from any other web pages, the “Product List” web page should not be sent to the client side.

Since the “Input Web Page Domain” for the “Shopping Cart” web page is {“Product List”}, only requests coming from the “Product List” web page should be processed. If requests come from any other web pages, the “Shopping Cart” web page should not be sent to the client side.

3.2 Cookies Support Detector

“Cookies Support Detector” technique is proposed to detect whether the client side supports cookies or not. Based on whether the client side supports cookies, the server side can adopt different session tracking technique to maintain session status.

The basic idea to detect whether the client side supports cookies is shown in Figure 3.2. There are four steps:

1. Create cookies.
2. Send cookies to the client’s side.
3. Get cookies from the client’s side.
4. Test the cookies' length. In implementation, cookies are an array of String data type objects since cookies are actually string of characters. Therefore, cookies do not exist means that the value of the String data type object is **null**. Thus, if the test result is null, we can conclude that the client side does not support cookies or the client side disables cookies. Notice the fact that some

web browsers do not support cookies and some web browsers provide the option that a user can disable cookies. If the cookies' length is greater than zero, we can conclude that the client's side supports cookies.

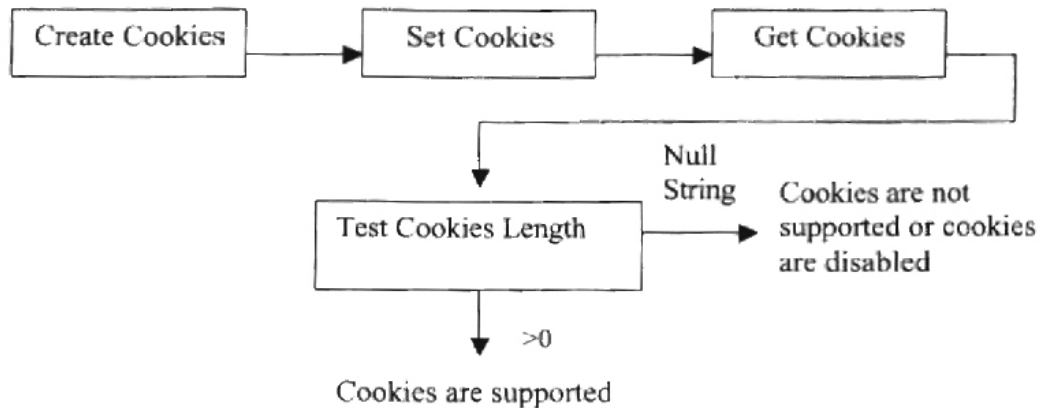


Figure 3.2 Cookies Support Detector

3.3 A Server-Side Security Model

Before introduce the proposed server-side security model, we first discuss the problems of existing security approaches used in web applications. We then propose the server-side security model that takes advantage of existing security approaches and combines them with the proposed two techniques: Input Web Page Domain Checking and Cookies Support Detector.

3.3.1 Problems of Existing Security Approaches

In Chapter 2, we reviewed existing security approaches and their advantages.

Now, we will discuss the problems of these existing security approaches.

Firewall

Firewall is widely used as the first layer protection for web application security. It is an electronic gate that limits access between networks in accordance with local security policy [Goldberg 2002]. Though firewalls can prevent illegitimate traffic from traveling from the Internet to the corporate networks, they do little to protect against inbound malicious requests for legitimate applications. Legitimate requests that pass through a firewall may be used for a data-driven attack on the networks or back-end systems [Garfinkel 1997]. "Data-driven attack is a form of attack that is encoded in innocuous seeming data which is executed by a user or other software to implement an attack. In the case of firewalls a data driven attack is a concern since it may get through the firewall in data form and launch an attack against a system behind the firewall." [Garfinkel 1997]

Secure Socket Layer (SSL)

Secure Socket Layer (SSL) is often deployed to prevent data theft from the user's browser to a web site. However, SSL is not used in all web sites although it can be easily implemented to make data more secure. SSL dramatically affects the speed at which users can access information, since a large amount of extra processing occurs in SSL due to encryption and decryption. Performance of an application is often decreased if SSL is used. Therefore, it is better to use SSL only when sending confidential data over the Internet. In a web application design, one can mix SSL (using http protocol) and non-SSL (using https protocol).

Intrusion Detection System (IDS)

Intrusion Detection is a technique that attempts to discover attacks, preferably to discover those attacks still under way. However, both pattern-based and anomaly-based IDS have significant drawbacks. Pattern-based IDS detects attacks with known patterns. Since only limited attack patterns are known, the effectiveness of pattern-based IDS is also limited. Anomaly-based IDS detects attacks when an unknown pattern is seen. Since the real time normal behaviors of the web application cannot be predicted completely, it is possible that the anomaly-based IDS cause authorized users not be able to gain access.

Cookies

Cookies technology is the easiest way to maintain session status. However, there are two disadvantages in using cookies. First, using cookies is not foolproof to those who know how to bypass the authentication. Cookies may allow a malicious user to hijack web sessions and view, modify, or exploit the information related to another user's session. A hacker may obtain cookies by various means, including physical access or network sniffing, as well as guessing the cookies contents. Then the hacker can impersonate the user by hijacking the user's sessions. If an unauthorized user is able to capture the cookies, he or she may be able to gain unauthorized access to personal information. Secondly, users who have cookies disabled will not be able to be authenticated.

Access Control

Access control is often used to allow authenticated users to perform certain operations they are authorized to. For those attacks bypassed authentication successfully,

access control exists in name only.

Session timeout

Session timeout is used to enforce users to re-authenticate if a preset amount of time is passed. However, it is difficult to set a perfect value for the session timeout: time too short is inconvenient to the user, too long may provide a chance for attacker.

3.3.2 Proposed Server-Side Security Model

In previous section, we discussed the problems of existing security models. In this section, we propose the server-side security model which takes advantage of existing security models and combines them with two new techniques: Input Web Page Domain Checking and Cookies Support Detector, to provide a secure protection for web applications. Figure 3.3 shows the model. The proposed server-side security model sets several security layers between the client's request and the server's response to prevent sensitive information from theft.

Layer 1: Sanitizing Browser Inputs

Purpose: Prevent "cross site scripting" attack (Chapter 2, section 2.9), which embed malicious HTML tags or special characters such as ! and & in the client web request to reveal sensitive information.

Layer 2: Cookies Support Detector

Purpose: Cookies Support Detector (Chapter 3, section 3.2) is used to detect whether the client's side supports cookies. The server side adopts different session

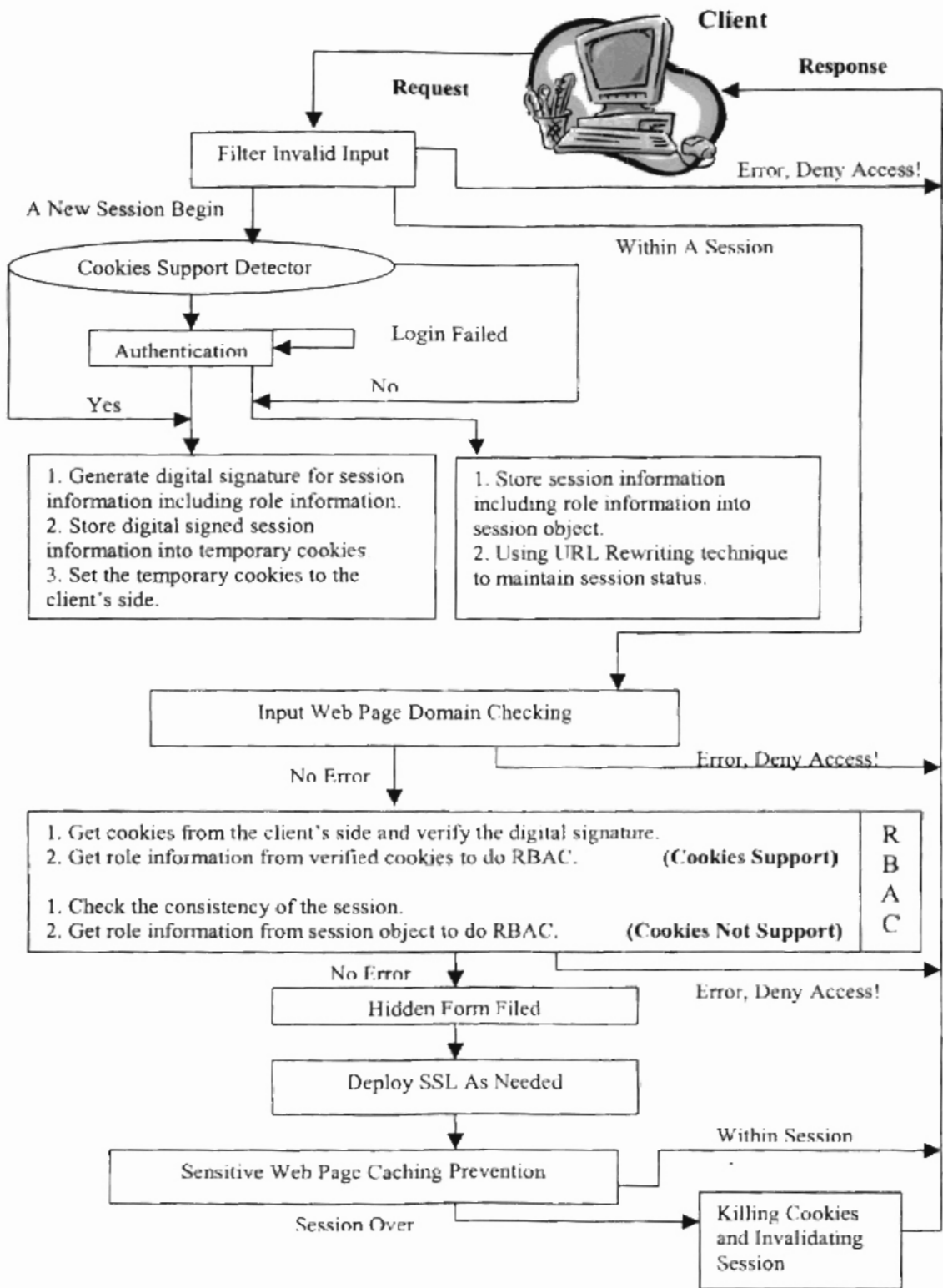


Figure 3.3 A Server-Side Security Model For Web Applications

tracking techniques based on the detection result.

Layer 3: Authentication

Purpose: Authentication is the traditional way to protect web applications and is used to ensure that the client is the person he said he is. Today's authentication standard is Username/Password authentication. It is also used in this server-side security model.

Layer 4: Setting Temporary Digital Signed Cookies

Purpose: Set digital signed temporary cookies to the client's web browser. If the client side supports cookies, after the client's authentication, the server side gets the client's role information from a database and stores them (possibly with other session information) into digital signed temporary cookies. Since cookies by themselves are insecure (Chapter 3, section 3.3.1), using digital signature algorithm to digitally signed cookies can prevent cookies from theft or modification. In this server-side security model, we adopted the technique of "persistent digital signed cookies" introduced in [Park 2001]. However, instead of using persistent cookies, we use temporary cookies to make the cookies more secure, since temporary cookies are deleted immediately after each session is over. We call the cookies "temporary digital signed cookies".

Layer 5: URL Rewriting

Purpose: If the client side does not support cookies, URL Rewriting (Chapter 2, section 2.5.1) is then applied. URL rewriting is a method in which the requested URL is modified to include a session ID. It works with browsers that do not support cookies, or when the client has disabled cookies.

Layer 6: Input Web Page Domain Checking

Purpose: Web applications are based on the client-server architecture. For each user request, the web application server sends corresponding web page from the server side to the client side. There are some particular orders (application workflow) when accessing web pages. The proposed Input Web Page Domain Checking technique is used to ensure the correct application workflow, and prevent malicious user from accessing sensitive information without authentication.

Layer 7: Role-Based Access Control

Purpose: Role-Based Access Control (RBAC) is the most widely used authorization technique. It is used to prevent malicious user (specifically, the legitimate user who can login but have no authorization to access certain web pages) to access sensitive information that is beyond his authorization.

Layer 8: Hidden Form Field

Purpose: Hidden form field (Chapter 2, section 2.5.2) is one of the simplest session-tracking techniques. Hidden form fields are HTML input types that are not displayed when read by a browser. This technique can prevent some malicious users to modify parameters displayed on a web browser to do some type of hacking, such as on-line shopping lifting or data-driven attack, etc.

Layer 9: Deploy Secure Socket Layer As Needed

Purpose: Secure Socket Layer (SSL, Chapter 2, section 2.8) technique provides some form of encryption to prevent data theft during the data transition. In web

application design, we can mix HTTP protocol with HTTPS protocol together. Since SSL dramatically affects the speed of web application performance, it is only recommend when sending a web page which contains sensitive information, such as user's credit card information, bank account information, etc.

Layer 10: Sensitive Web Page Caching Prevention

Purpose: Web browsers usually store a local copy of every web page a user visited to speeds up the access to Web pages. Preventing sensitive web page to be cached by client's web browser prevents malicious user accessing those sensitive web pages through clicking the "Back" button in the web browser to view the sensitive information ever after a client has logged off.

Layer 11: Killing cookies and Invalidating Session

Purpose: This is the last security procedure before a session is over. Since the cookies are temporary cookies in the proposed security model, cookies must be killed to end a session. Session is also to be invalidated to prevent malicious users accessing other people's session even after the other people has logged off.

To demonstrate the proposed server-side security model, a Wheat Bin Mix Optimization web application is designed and implemented. In Chapter 4, we will present in detail about wheat bin mix optimization web application. In Chapter 5, we will discuss how to build the proposed server-side security model into wheat bin mix optimization web application.

CHAPTER 4

Wheat Bin Mix Optimization Web Application

In this chapter, we introduce a Wheat Bin Mix Optimization web application. There are 4 sections in this chapter. Section 4.1 describes the objective of wheat bin mix optimization web application. Section 4.2 presents some background information about wheat bin mix optimization, including the discount table and the grade table used to calculate the discount. Section 4.3 shows the user interface and explains the function of each web page in the web application. Section 4.4 discusses the architecture of this web application; tools used for performing the wheat bin mix optimization; and the optimization algorithm.

4.1 Objective

The objective of wheat bin mix optimization web application is to maximize the profit by optimal blending of wheat stored in two or more bins at wheat elevators. The buying company will discount the seller's wheat according to the wheat grade standard, which is regulated by U.S. Department of Agriculture, as well as the buying company's own discount table. This web application is used to help elevator managers finding optimal blending strategy to maximize their profits.

4.2 Background Information

In this section, we first introduce some terminologies related to wheat bin blending, standard wheat grade table, and discount table. Later, a sketch of wheat bin mix optimization web application will be presented. Finally, limitations of the optimization algorithm and the expandability of wheat bin mix optimization web application will be discussed.

4.2.1 Terminologies

The following terminologies are related to wheat quality. They also appear as the column name in the bin information table of wheat bin mix optimization web application.

- *Bushels*: the volume unit of a wheat bin.
- *Total Height (ft)*: the height of a wheat bin.
- *Head Space (ft)*: the height of the empty space in the bin.
- *Break Point (ft)*: the height of the filled space in the bin.
- *Moisture (%)*: moisture weight percentage, an essential measure of wheat's storability and value.
- *Test Weight (LB)*: test weight per bushel. It is the weight of the volume of grain that is required to fill a Winchester bushel (2,150.42 cubic inch) to capacity.
- *Dockage (%)*: material other than the predominant grain that can be easily removed with sieves and cleaning devices.

- *SBK (%)*: shrunken and broken kernels, all matter that pass through a 0.064-inch by 3/8-inch oblong-hole sieve.
- *FM (%)*: foreign material. All matter other than wheat that remains in the sample after the removal of dockage and shrunken and broken kernels.
- *HDK (%)*: heat damaged kernels, kernels that are materially discolored and damaged by external heat or as the result of heating caused by fermentation.
- *IDK (%)*: insect-damaged kernels, kernels that bear evidence of boring or tunneling by insects.
- *Damage (%)*: total damaged kernels, kernels that include weather-damaged, heat-damaged and insect-damaged etc.
- *Defect (%)*: total amount of damaged kernels, foreign material, and shrunken and broken kernels.
- *WCC (%)*: wheat of contrasting classes, which are: Durum wheat, hard white wheat, soft white wheat and unclassified wheat.
- *WOC (%)*: wheat of other classes, classes other than the contrasting classes which including hard red spring wheat, hard red winter wheat and mixed wheat.
- *Protein (%)*: the weight percentage of protein contained in wheat.

To perform bin-blending optimization, the value of “Bushels” must be given by a user. Otherwise it is treated as 0, which indicates an empty bin. There is no reason for blending an empty bin. Other values, if missing, are assumed to have the best value, i.e., no discount is to be counted for missing values. For example, if the test weight value is

missing, it is assumed to be equal to 60 lb. However, at least one other value should be given besides the “bushels”.

4.2.2 Standard Wheat Grade Table

In wheat bin mix optimization web application, the following wheat grade table is used. This is the standard grade table published by Federal Grain Inspection Service in February 2002 [Federal Grain Inspection Service 2002].

Table 4.1: 810.2204 Grade and Grade Requirements for Wheat

Grade	TW (lb)	HDK (%)	Damage (%)	FM (%)	SBK (%)	Defect (%)	WCC (%)
1	60	0.2	2	0.4	3	3	1
2	58	0.2	4	0.7	5	5	2
3	56	0.5	7	1.3	8	8	10
4	54	1	10	3	12	12	10
5	51	3	15	5	20	20	10

In wheat grading, Grade 1 is the highest grade, and Grade 5 is the lowest grade. The following indices are considered for grading the wheat: Test Weight (TW), Heat Damaged Kernel (HDK), Damage, Foreign Material (FM), Shrunken Broken Kernel (SBK), Defect, and Wheat of Contrasting Class (WCC). The grading procedure can be described as follows:

- 1) For given wheat, get the grade based on each single index. The grade should be one grade lower than the grade whose index value is just greater than the index value for that wheat. For example, if the TW value of given wheat is 57 lb, we should classify the grade to 3 based only on the TW.

2) Take the lowest grade from step 1 to be the final grade.

For example, a user gives the TW value and HDK value of wheat in a bin as 59.9 lb and 1.2%, respectively. Based on the TW value, the grade is 2; based on the HDK value, the grade is 5. Therefore, the final grade is the lower of 2 and 5, which is grade 5.

4.2.3 Discount Table

The Peavey Company (5301 West Channel Road, Catoosa, Oklahoma, 74015) has a discount table effective as of June 2000. We took this discount table as a “standard” discount table. The discount table can be modified and saved for future use (but should be in the same format in the web application design) if a buying company has a different discount table other than the Peavey Company.

The following is the “standard” discount table used in the web application. In this table, negative values represent discount and positive values represent premium [Peavey Company 2000]. All values are per bushel value.

Table 4.2: Discount Table (Peavey Company)

Index	>	<	Cents
Grade	1	1	0
	2	2	-0.5
	3	3	-3
	4	4	-6
	5	5	-9
	sample		-12
	*****	*****	*****
Moisture (%)	13.5	0	

	13.6	13.7	-2
	13.8	14.0	-4
	14.1	14.2	-6
	14.3	14.5	-8
	14.6		
	Each	0.25	-2
	*****	*****	*****
FM (%)		1.0	0
	1.1	5.0	
	Each	0.5	-1
	5.1	10.0	
	Each	1	-5
	*****	*****	*****
TW (lb)	58.0		0
	55.0	57.9	
	Each	0.5	-2
	54.0	54.9	
	Each	0.5	-4
	*****	*****	*****
WOC (%)		5.0	0
	5.1	10.0	
	Each	1	-5
	*****	*****	*****
Dockage (%)		1.0	0
	1.1	2.0	-2
	2.1	3.0	-4
	3.1	10.0	
	Each	0.5	-2
	*****	*****	*****

Damage (%)	3.0	0
3.1	10.0	
Each	1	-1
10.1	15.0	
Each	1	-2

Protein (%)	12.0	6
11.5	11.9	6
10.5	11.4	3
10.0	10.4	-5
	9.9	-10

For example, to get the discount related to moisture (moisture % is rounded to one decimal point) of a bin of wheat, the table can be read as follows (others are similar):

- 1) If moisture is less than or equal to 13.5%, the wheat is discounted by 0 cent per pound (no discount).
- 2) If moisture is either 13.6% or 13.7% (notice that only moisture percentage value has only one decimal point), the wheat is discounted by 2 cents per pound.
- 3) If moisture is between 13.8% and 14.0%, the wheat is discounted by 4 cents per pound.
- 4) If moisture is either 14.1% or 14.2%, the wheat is discounted by 6 cents per pound.
- 5) If the wheat's moisture is between 14.3% and 14.5%, the wheat is discounted by 8 cents per pound.

- 6) If moisture is greater than or equals to 14.6%, then the wheat is discounted by 8 cents discount, plus extra 2 cents discount per 0.25% over 14.5%, per pound.

4.2.4 Sketch of Wheat Bin Mix Optimization Web Application

The implementation of this web application was divided into two phases: User interface construction and back-end optimization algorithm implementation.

In phase I, the user interface was constructed. When a user accesses the home page of the web application, the web application prompts the user to input number of bins. After the number of wheat bins are selected from the drop-down menu, a *bin information* web page is displayed, asking the user to provide wheat bin information, which are the values of some or all index of each wheat bin, such as bushels, moisture and protein. The bin information can also be loaded from an existing file in user's local disk.

A user can view the standard grade table by clicking the "Grade Table" button, and view the discount table by clicking the "Discount Table" button. After the user submits the wheat bin information, the grade and the discount of the wheat in each bin can be calculated by clicking the "Calculate Grade And Discount" button. Based on the values of the grade and discount, the user can select some of the wheat bins for optimal blending by clicking the "Optimization" button. By clicking the "Calculate Rank" button, the user can get the rank of the bins based on variables the user selected, such as moisture, discount, protein, and test weight.

In phase II, the back-end optimization algorithm was implemented using Java language. The algorithm is a pattern search method [Lewis 2000] proposed by Lewis, Torczon, and Trosset. Since this algorithm performs a random search, it is not guaranteed that the optimal point is the global optimal point. The user can perform the optimization several times and record all plausible results. Then the user can select one best possible blending option to perform the blending. The user's final choice may not be the "true" minimum discount blending, since the user may choose the one that requires the minimum steps of blending. For example, suppose the user has two options: the first option requires 4 steps of blending with discount \$11,010; the second option requires 3 steps of blending with discount \$11,000. In terms of discount, \$10 difference in \$11,000 is negligible. Therefore, the user may select the second option.

4.2.5 A Limitation of The Optimization Algorithm

The back-end optimization algorithm — **pattern search** method is "characterized by a series of exploratory moves that consider the behavior of the objective function at a pattern of points, all of which lie on a rational lattice" [Lewis 2000]. Due to the characteristics of the pattern search method, we set the maximum number of candidate wheat bins for optimization to be 5. This is equivalent to say that we are searching in a maximum 5 dimensional space. Otherwise, the speed of the optimization will be too slow for practical use. It is obvious that the minimum number of wheat bins chosen for optimization should be 2. Otherwise there is no blending occurs.

4.2.6 Expansion Capability

The wheat bin mix optimization application only applies to **wheat**. However, it can also be used for other products, such as corn, soybean, provided that we use the related grade table and discount table.

4.3 User Interface

This web application consists of the following web pages and pop-up windows:

- Web pages
 - Home
 - Bin Information
 - List Bin Information
 - List Bin From A File
 - Grade and Discount Calculation
- Pop-up Windows
 - Optimization Setting
 - Optimization Setting Error
 - Optimization Result
 - Rank Criteria Setting
 - Rank
 - Grade Table
 - Discount Table

The function of each web page and pop-up window will be discussed in detail as follows:

- **Home web page (Figure 4.1):** displays a drop-down window for users to choose the maximum number of candidate wheat bins for optimization. We limit the minimum number to be 2 instead of 1, since there is no meaning to mix only 1 candidate bin. We also limit the maximum number to be 30. Home web page retrieves the number of wheat bin chosen by a user and passes it to the **Bin Information** web page.



Figure 4.1 Home Web Page

- **Bin Information** web page (Figure 4.2): displays a wheat-bin-information table and asks users to provide wheat bin information. Users can upload wheat bin information from an existing file or manually enter wheat bin information in the

wheat-bin-information table. If a user manually enters the wheat bin information, the wheat bin information can be saved in a file to the user's local hard disk for future use. Users can calculate grade and discount for each wheat bin by clicking the button "Calculate Grade & Discount". In this web page, users can also view the standard grade table and the standard discount table by clicking the "Grade Table" button and "Discount Table" button. If a user manually enters wheat bin information, Bin Information web page retrieves those data and passes them to the List Bin Information web page. Otherwise, it just passes the directory of a file given by users to the List Bin From A File web page.

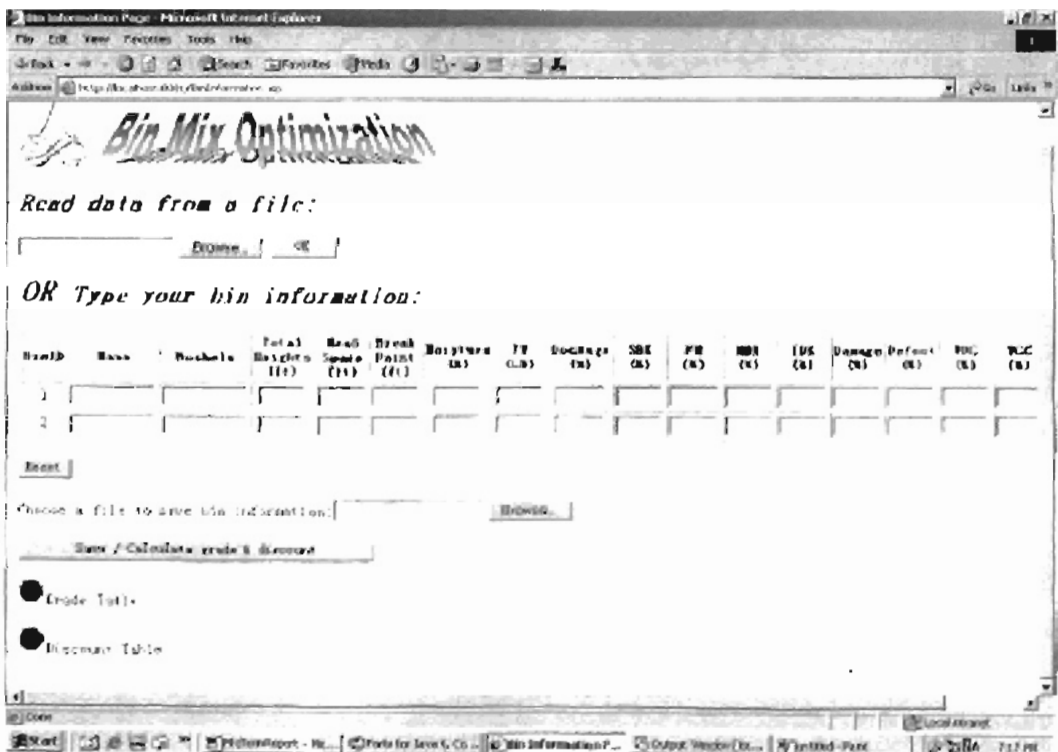


Figure 4.2 Bin Information Web Page

- **List Bin Information** web page (Figure 4.3): gets manually entered wheat bin information passed from the Bin Information web page and calculate both grade and discount for each wheat bin. List Bin Information web page also displays the manually entered wheat bin information and their grades and discount in a table. Users can do optimization by clicking the “Optimization” button in this web page. The standard grade table and discount table can also be viewed on this web page by clicking the “Grade Table” button and the “Discount Table” button.

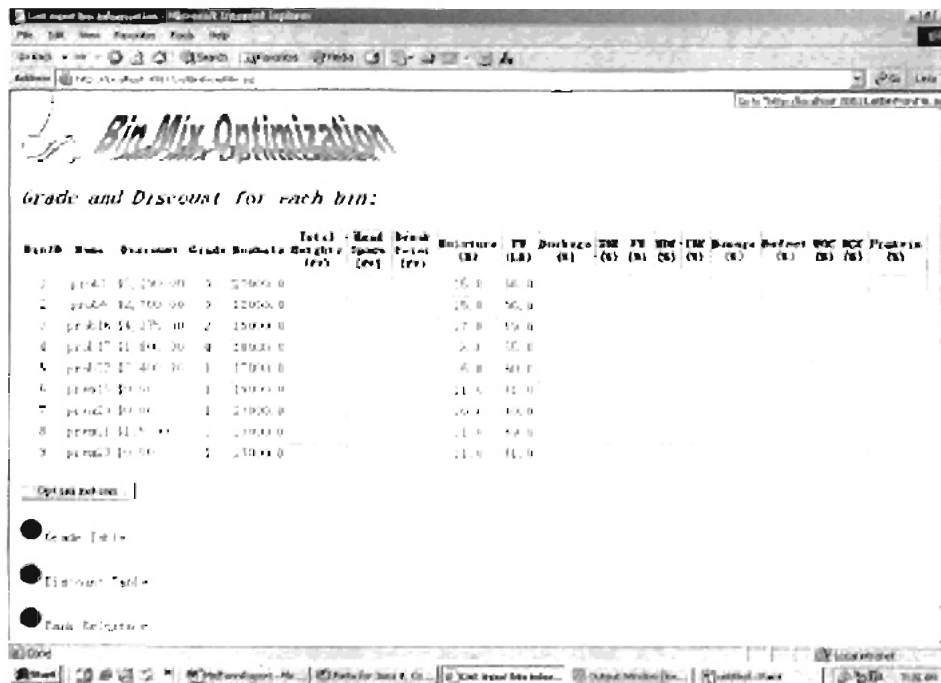


Figure 4.3 List Bin Information Web Page

- **List Bin Information From A File** web page (Figure 4.4): gets the directory of a file passed from the Bin Information web page and opens the file. List Bin Information From A File web page also retrieves the data (wheat bin information)

in that file and displays them in a table, and provides “Calculate Grade and Discount”, “Grade Table” and “Discount Table” buttons.

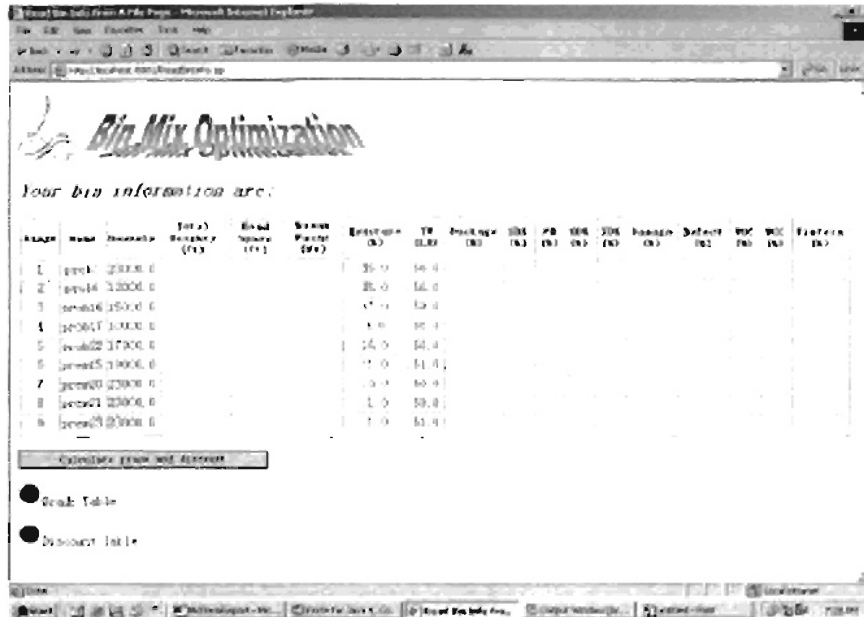


Figure 4.4 List Bin Information From A File Web Page

- **Optimization Setting** pop-up window (Figure 4.5): displays a table for optimization settings. In this pop-up window, users must type the original wheat bin IDs (candidate bins) and the destination wheat bin IDs (put the mixed wheat after optimization). The maximum number of candidate wheat bins chosen for optimization is 5. Otherwise the speed of the optimization will be too slow for practical use. The minimum number of wheat bins chosen for optimization is 2, because it makes no sense if there is only 1 bin for blending. The number of the original bin should equal to the number of the destination bin (just because of the program design). If the two numbers are not equal, the “Optimization setting error” pop-up window is displayed to show the error message.

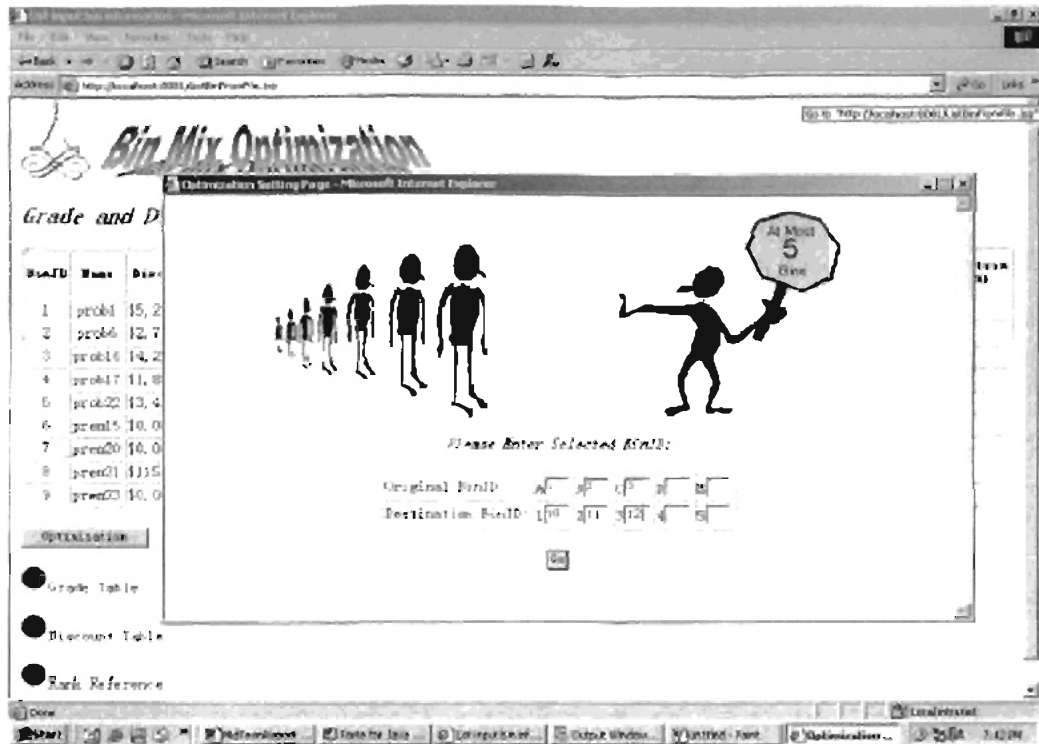


Figure 4.5 Optimization Setting Pop-up Window

- **Optimization Setting Error pop-up window** (Figure 4.6): displays the error message if one or more of the following errors occur:
 - A user selected zero or one wheat bin for optimization;
 - The number of the original bin doesn't equal to the number of the destination bin;
 - Duplicate original wheat bin ID or destination wheat bin ID;

Optimization Setting Error pop-up window also provides a “Redo” button so that a user can redo the optimization setting.

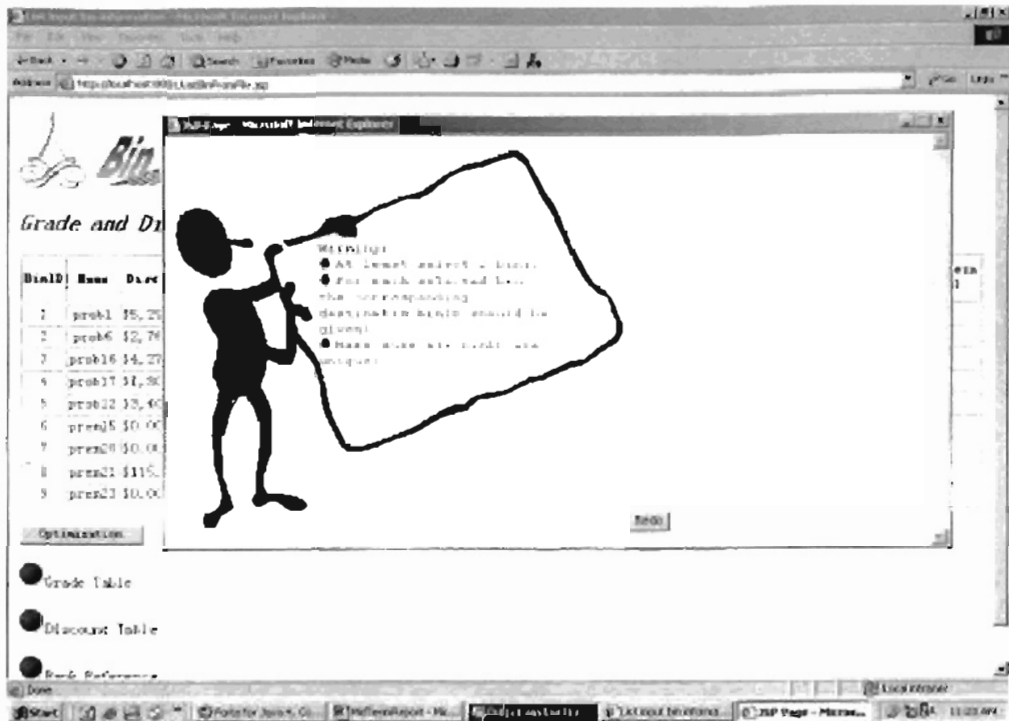


Figure 4.6 Optimization Setting Error Pop-Up Window

- Optimization Result** pop-up window (Figure 4.7): displays the optimization result. The optimization result tells the user how to blend the wheat from each original wheat bin into a destination wheat bin, i.e., how many bushels of wheat should be taken out from each one of the original wheat bin to a destination wheat bin. The Optimization Result pop-up window also displays the total bushels, discount, grade of each destination wheat bin as well as the total discount after the blending. Optimization Result pop-up window provides a “Record Optimization Result” button by which a user can save the optimization result to a file for later use. Users can choose to append or overwrite the optimization result to a file by “Append” or “Overwrite” button.

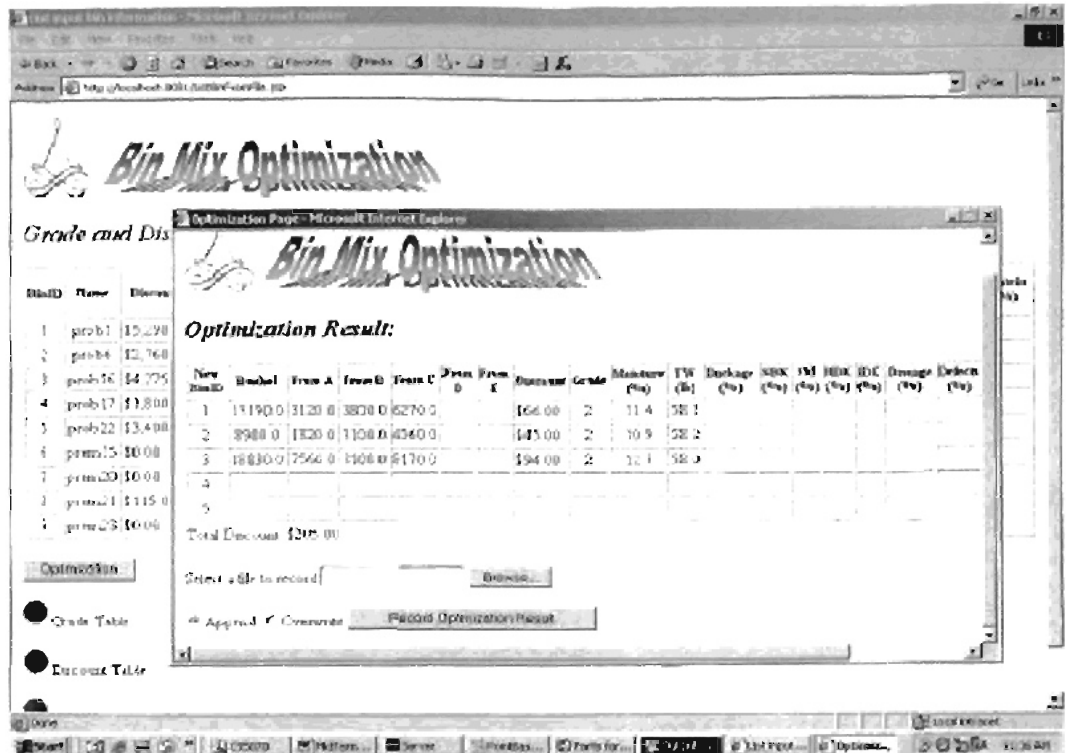


Figure 4.7 Optimization Result Pop-Up Window

- **Rank Criteria Setting** pop-up window (Figure 4.8). displays criteria for calculating rank of each wheat bin. The criteria are discount, moisture, test weight, dockage and protein. Discount is selected by default.
- **Rank** pop-up window (Figure 4.9): displays the rank value for each wheat bin based on the user's criteria selection.
- **Grade Table** pop-up window (Figure 4.10): displays the Federal Standard Grade Table as shown in Table 4.1.
- **Discount Table** pop-up window (Figure 4.11): displays the discount table from Peavey Company, as shown in Table 2.



Figure 4.8 Rank Criteria Setting Pop-Up Window

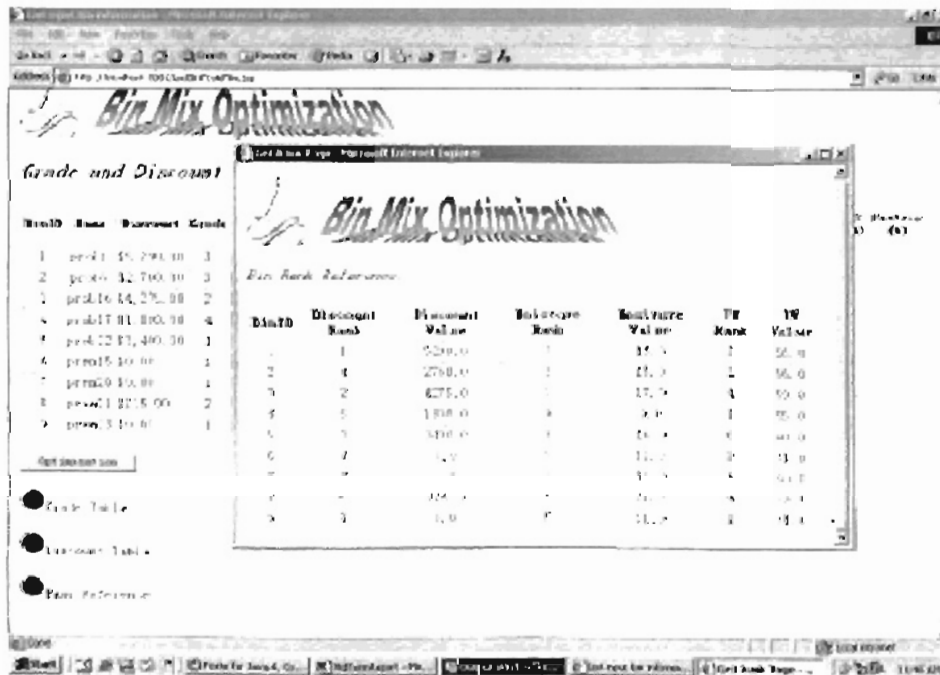


Figure 4.9 Rank Pop-Up Window

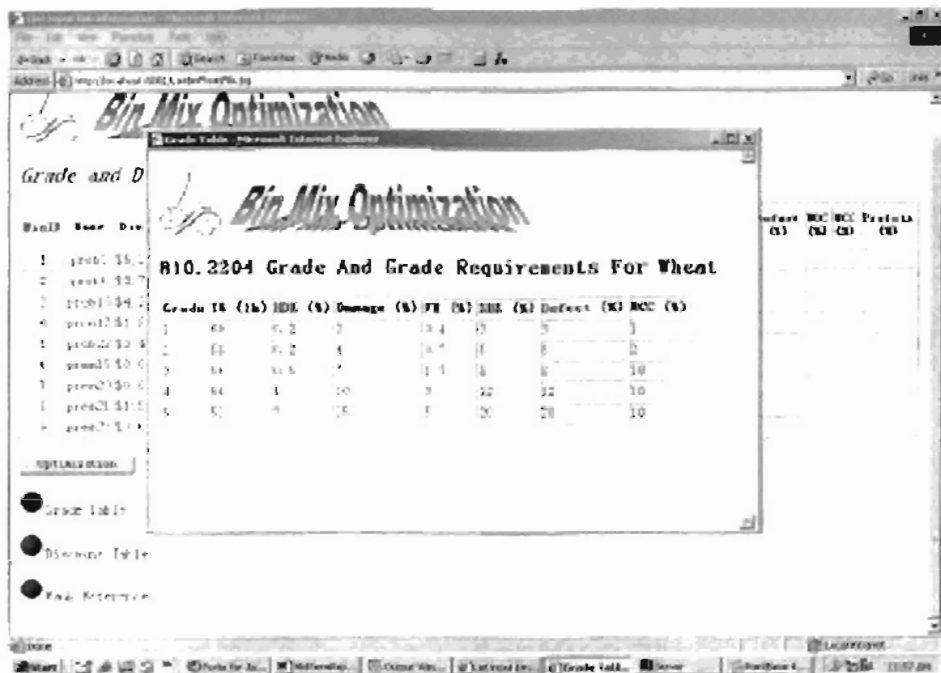


Figure 4.10 Grade Table Pop-Up Window

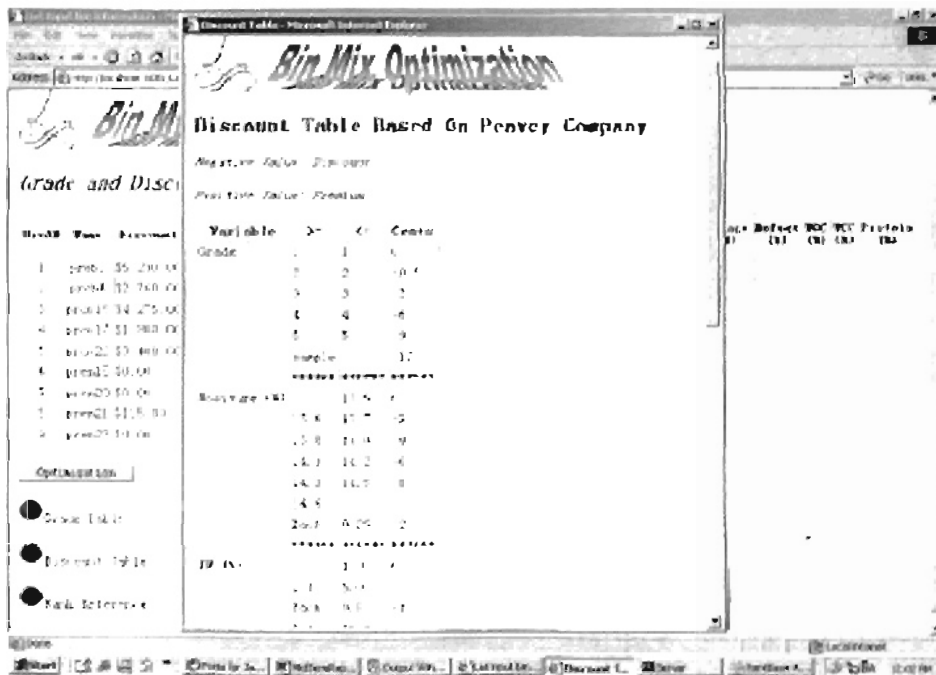


Figure 4.11 Discount Table Pop-Up Window

4.4 Inside of Wheat Bin Mix Optimization Web Application

In this section, we first present the architecture of wheat bin mix optimization web application and tools used to build this web application. Then we introduce the basic idea to implement the pattern search optimization algorithm.

4.4.1 Architecture

Figure 4.12 shows the architecture of wheat bin mix optimization web application.

Briefly, the elements shown in Figure 4.12 are:

- The *client* component

The client component is a web browser that displays the application pages.

- The *service* component, Java Server Pages (JSP), includes:
 - A **HomePage** JSP page
 - A **BinInformation** JSP page
 - A **ListBinInfo** JSP page
 - A **ReadBinFromFile** JSP page
 - A **ListBinFromFile** JSP page
 - A **OptiSetting** JSP page
 - A **Optimization** JSP page
 - A **CriterialSet** JSP page
 - A **GetRank** JSP page
 - A **Discount** JSP page
 - A **GradeTable** JSP page

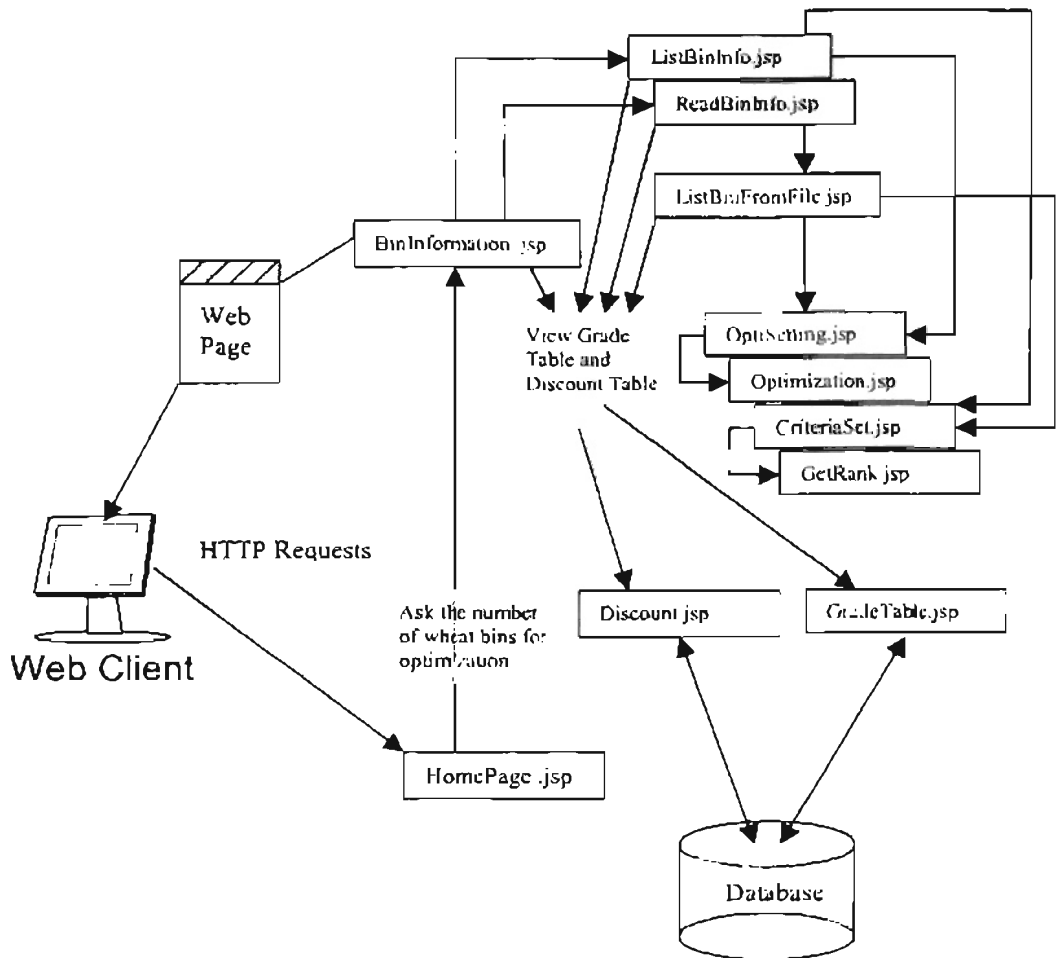


Figure 4.12 The Architecture of Wheat Bin Mix Optimization Web Application

Each service component is used to create different user interface and implement different functions. We already introduced in detail about the user interface and functions of wheat bin mix optimization web application in section 4.3. The corresponding relationship between each service component and each user interface is listed in Table 4.3.

Table 4.3: Mapping Each Service Component to Each User Interface

Service Component	Corresponding User Interface and Function
HomePage.jsp	Home web Page (section 4.3)
BinInformation.jsp	Bin Information web page (section 4.3)
ListBinInfo.jsp	List Bin Information web page (section 4.3)
ReadBinFromFile.jsp	No corresponding user interface. It is just used to open a file and pass the data of a file to the ListBinFromFile JSP page.
ListBinFromFile.jsp	List Bin Information From A File web page
OptiSetting.jsp	Optimization Setting pop-up window (section 4.3)
Optimization.jsp	Optimization Result pop-up window (section 4.3)
CriteriaSet.jsp	Rank Criteria Set pop-up window (section 4.3)
GetRank.jsp	Rank pop-up window (section 4.3)
Discount.jsp	Discount Table pop-up window (section 4.3)
GradeTable.jsp	Grade Table pop-up window (section 4.3)

4.4.2 Tools Used

The *database server* used in the web application is Sun PointBase 4.2 [Sun Microsystem 2002]. Database server is used to support database in a web application. There are two databascs created for wheat bin mix optimization web application. One is the standard wheat grade table [Federal Grain Inspection Service 2002] provided by the U.S. Department of Agriculture. The other is the discount table [Peavey Company 2000] provided by the Peavey Company.

Tomcat 4.0 [Sun Microsystem 2002], a *web application container*, makes the wheat bin mix optimization web application accessible from the web.

4.4.3 Pattern Search Optimization Algorithm

The back-end optimization algorithm--Pattern Search algorithm, is implemented in Java language. For details about the Pattern Search algorithm, see [Lewis 2000]. The pattern search algorithm is implemented according the following steps (for simplicity, we assume there is only one variable x in the objective function):

- Initialization: initialize x_1 to random value.
- Search and Poll: at iteration x_t , evaluate the objective function at a finite number of points on a mesh to find one that yields the lowest objective function value.
- Parameter Update: refine the mesh, setting $x_{t+1} = x_t$, do search step again.
- Stop: until no non-increasing function value is found.

In Chapter 5, we will discuss how to build the proposed server-side security model described in Chapter 3 into wheat bin mix optimization web application.

CHAPTER 5

Security Model in Wheat Bin Mix Optimization Web

Application

In Chapter 4, we presented the architecture of the “bare” wheat bin mix optimization web application, which was not protected by any security measures. Suppose wheat bin mix optimization web application is a member-only web application, i.e., each user of this web application must have the membership of wheat bin mix optimization web site. How can we prevent unauthenticated users from accessing this web application? How can we prevent unauthorized users from changing the default grade table or discount table? The answer to those questions is to build the server-side security model proposed in Chapter 3 into the “bare” wheat bin mix optimization web application. We call the resulting web application as **protected** wheat bin mix optimization web application.

In this chapter, we will demonstrate how the server-side security model can be implemented by building this model into wheat bin mix optimization web application presented in Chapter 4. We will discuss in detail about how to implement each security layer of the server-side security model.

5.1 Architecture

The architecture of the **protected** wheat bin mix optimization web application is shown in Figure 5.1.

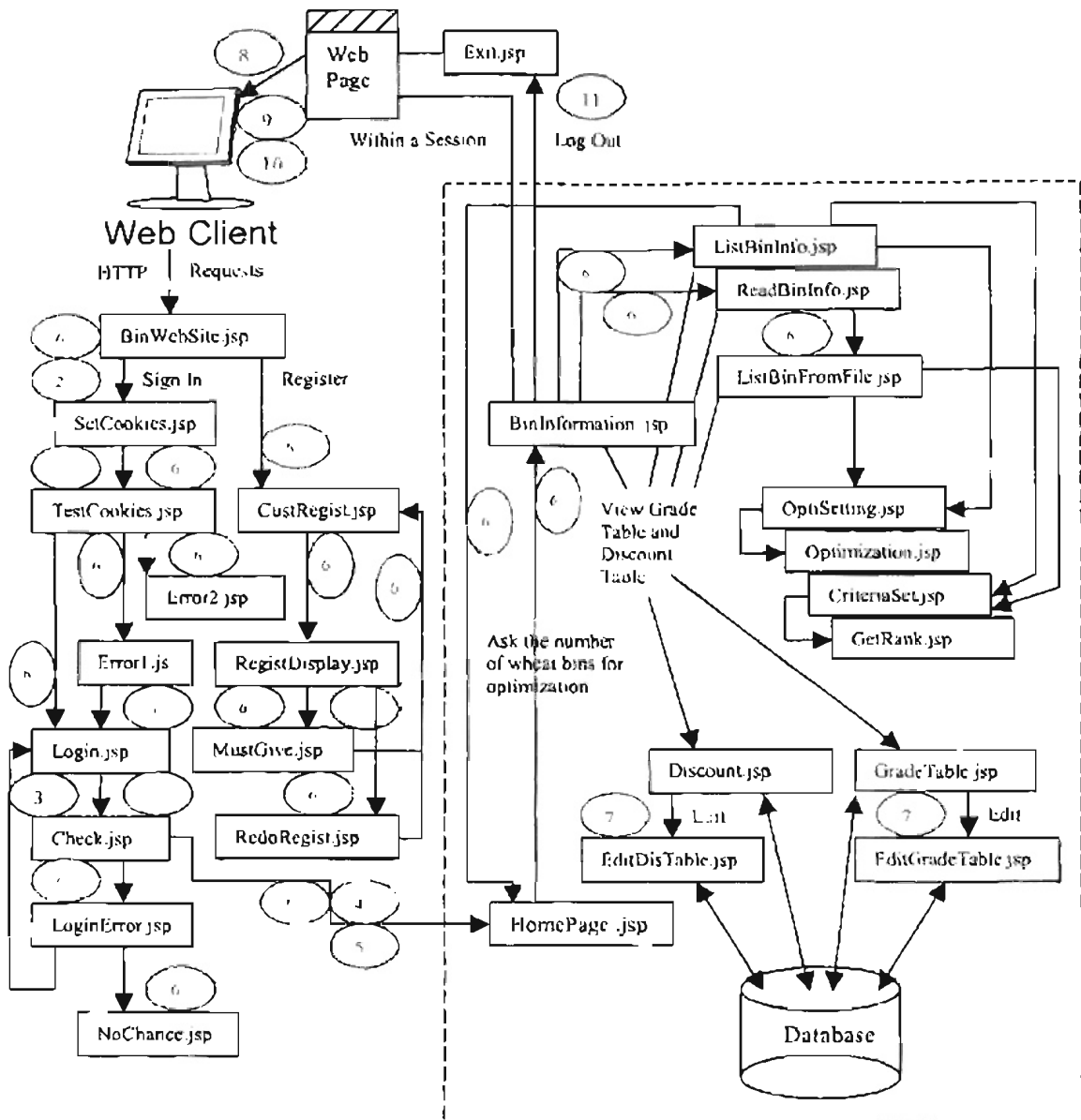


Figure 5.1 The Architecture of Protected Wheat Bin Mix Optimization Web Application

In Figure 5.1, number 1 to 11 with circles is used to denote the proposed eleven security layers in Chapter 3 respectively:

- Layer 1--Sanitizing Browser Inputs

- Layer 2--Cookies Support Detector
- Layer 3--Authentication
- Layer 4--Set Digital Signed Temporary Cookies
- Layer 5-- URL Rewriting
- Layer 6--Input Web Page Domain Checking
- Layer 7--Role Based Access Control
- Layer 8--Hidden Form Field
- Layer 9--Deploy Secure Socket Layer as Needed
- Layer 10-- Sensitive Web Page Caching Prevention
- Layer 11--Killing cookies and Invalidating Session

Compare to the architecture of the “bare” wheat bin mix optimization web application (Figure 4.12), some new web pages and pop-up windows are added in Figure 5.1. Those new service components are: “BinWebSite”, “SetCookies”, “TestCookies”, “CustRegist”, “Error1”, “Error2”, “RegistDisplay”, “Login”, “MustGive”, “Check”, “RedoRegist”, “LoginError”, “NoChance”, “EditDisTable”, “EditGradeTable”, and “Exit” web pages. Those web pages are written in Java Server Pages (JSP). We will introduce those service components in detail in section 5.2.

5.2 New Service Components

- **BinWebSite** Web Page (Figure 5.2)

This page is the home page of the protected wheat bin mix optimization web application. Each time a user access the wheat bin mix optimization web site, this page is the first web page sent to the user. It gives users several options.

These operations are “Sign In”, “Register”, “Introduction”, “Optimization”, “Objective”, and “Contact Us”. If a user is a member of this web site, he or she may access this web site by clicking the “Sign In” button. If a user is not a member of this web site, he or she can click the “Register” button to register first. The “Introduction” button provides a brief introduction to the wheat bin mix optimization web application. The “Optimization” button gives a description of the optimization algorithm. The “Objective” button describes the objectives of wheat bin mix optimization web application. If users have

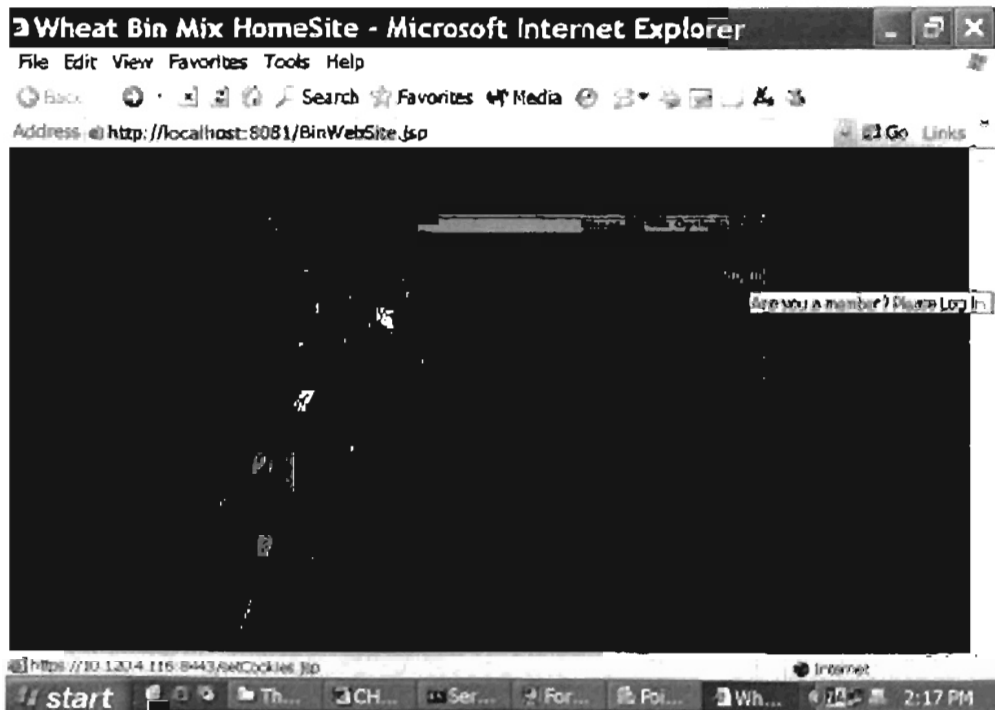


Figure 5.2 BinWebSite Web Page

any suggestions or problems about the wheat bin mix optimization web application, they can send email to wheat bin mix optimization web site by pressing “Contract U’s” button.

- **SetCookies Web Page (Figure 5.3)**

After a user clicks the “Sign In” button on the “BinWebSite” page, the server side sends “SetCookies” page to the user. This page provides users a button “Cookies Support Detector” through which the server side can test whether the user’s browser (the client’s side) supports cookies.



Figure 5.3 SctCookies Web Page

- **TestCookies Web Page (Figure 5.4)**

After a user clicks the “Cookies Support Detector” button in the “SetCookies” page, the server side detects whether the user’s browser supports cookies. If cookies are supported, the server side sends the TestCookies page to the user

as shown in Figure 5.4. Otherwise, the server side sends the Error1 page (Figure 5.6) to the user.

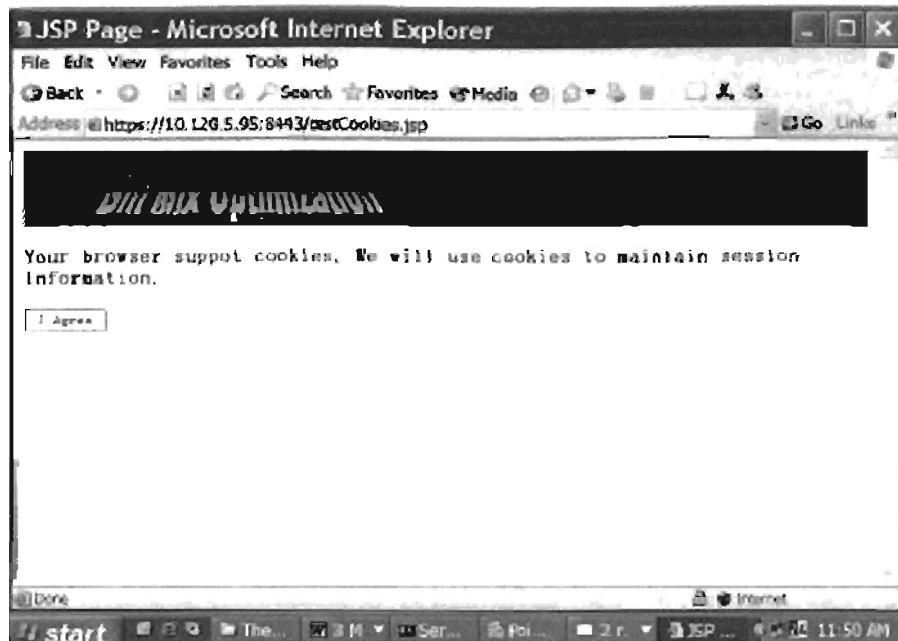


Figure 5.4 TestCookies Web Page

- **Error2 Web Page (Figure 5.5)**

If a user's browser supports cookies, but the server side can't get the exact cookies it sent to the user's browser, then the server side sends the Error2 JSP page to the user.

- **Error1 Web Page (Figure 5.6)**

If cookies are not supported or cookies are disabled in a user's browser, the server side sends the Error1 JSP page to the user. This page provides users a button "I Agree" By clicking this button, users may agree the server side to use session object to maintain session state.



Figure 5.5 Error2 Web Page

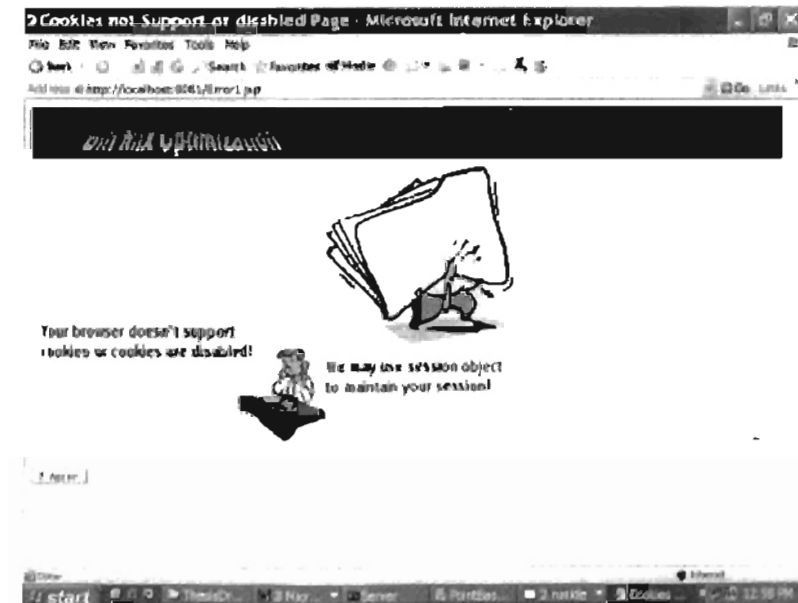


Figure 5.6 Error1 Web Page

- **Login Web Page (Figure 5.7)**

If a user's browser supports cookies, the server side sends TestCookies page to the user as shown in Figure 5.4. If cookies are not supported in the user's browser, the server side sends the Error1 page to the user as shown in Figure 5.6. After the user clicks the "I Agree" button provided in the TestCookies page or in the Error1 page, the server side sends the Login page to the user.

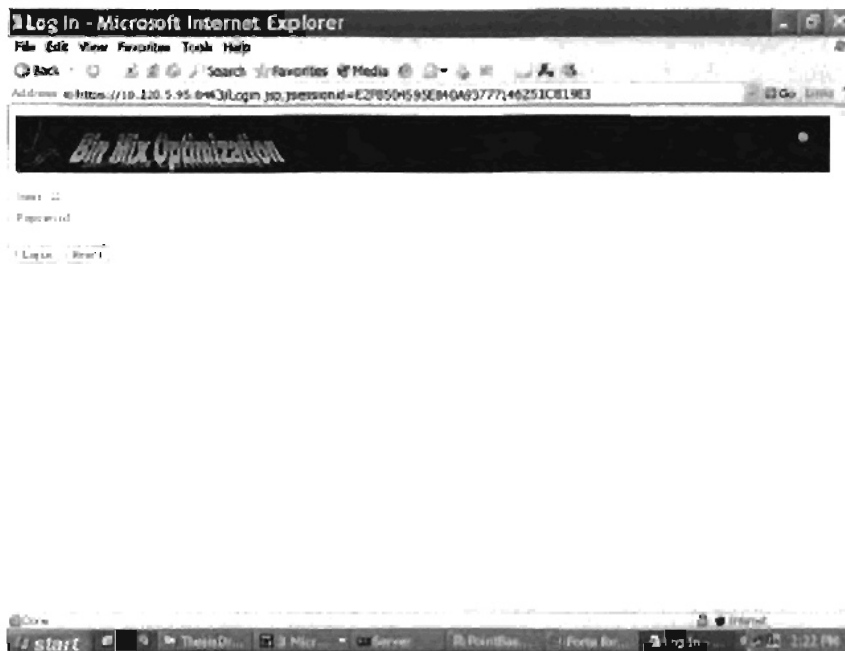


Figure 5.7 Login Web Page

- **Check Web Page (Figure 5.8)**

After a user submits username and password in the Login page, the server side compares the user's input username and password with records in its database. If the username / password pair is correct, the server side sends the Check

page to the user. This page provides users “Do Optimization” button. By clicking this button, the server side sends Home web page (Chapter 4) to users.

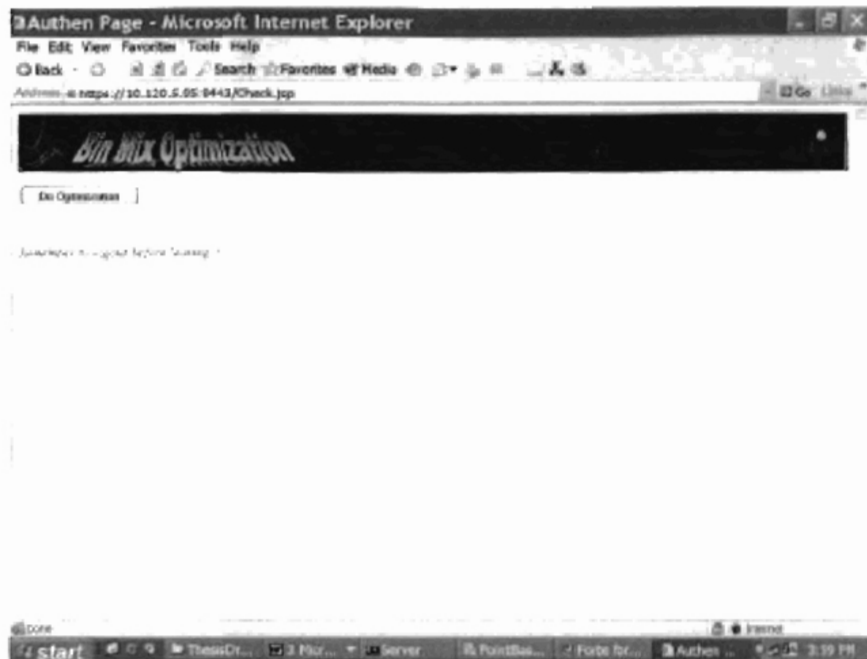


Figure 5.8 Check Web Page

- **LoginError** Web Page (Figure 5.9)

If a user submits incorrect username or password, the server side sends the LoginError page to the user. A maximum of three times retry is allowed.

- **NoChance** Web Page (Figure 5.10)

If a user provides the incorrect username / password pair for more than three times, the server side sends the NoChance page to the user.



Figure 5.9 LoginError Web Page

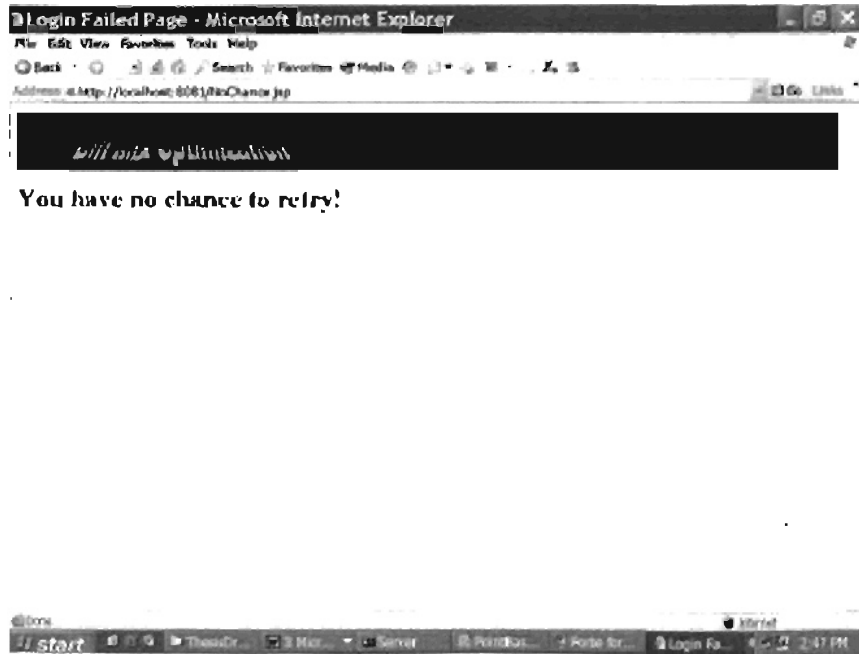


Figure 5.10 NoChance Web Page

- **CustRegist Web Page (Figure 5.11)**

If a user is not a member of the wheat bin mix optimization web site, he or she needs to register before accessing wheat bin mix optimization web application. The CustRegist page provides users a registration form. A user can click the “Submit” button on this page to submit registration information.



Figure 5.11 CustRegist Web Page

- **RegistDisplay Web Page (Figure 5.12)**

After a user submits registration information in the CustRegist page, the server side sends the RegistDisplay page to the user. This page displays users' registration information.

- **MustGive Web Page (Figure 5.13)**

- **RedoRegist Web Page (Figure 5.14)**

The security layer 1 “Sanitizing Browser Inputs” (Chapter 3, section 3.3.2) of the proposed security model is embedded in the RedoRegist page. If a user’s inputs contain any invalid characters such as !, \$, etc (Chapter 2, section 2.9), the server side detects those character from the user’s input and sends RedoRegist page to the user. We will explain in detail about how to implement “Sanitizing Browser Input” later in section 5.3.2.

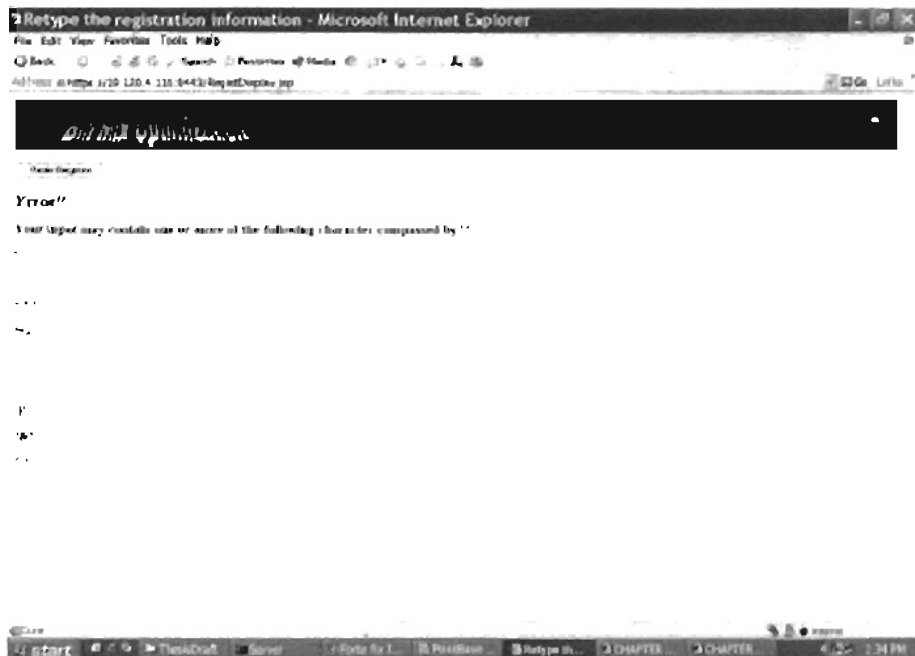


Figure 5.14 RedoRegist Web Page

- **EditDisTable Pop-up Window (Figure 5.15)**

The security layer 7 “Role Based Access Control” (Chapter 3, section 3.3.2) of the proposed security model is embedded in the EditDisTable pop-up

window. If a user plays a **role** (Chapter 2, section 2.4) as a System Administrator or a Web Application Designer, after he logs in, the server side gets his role from the back-end support database and sends the Discount Table pop-up window (Chapter 4) with an “Edit” button. By pressing the “Edit” button, the user can edit discount table in the EditDisTable pop-up window. We will explain how to implement the Role Based Access Control technique in wheat bin mix optimization web application later in section 5.3.2.

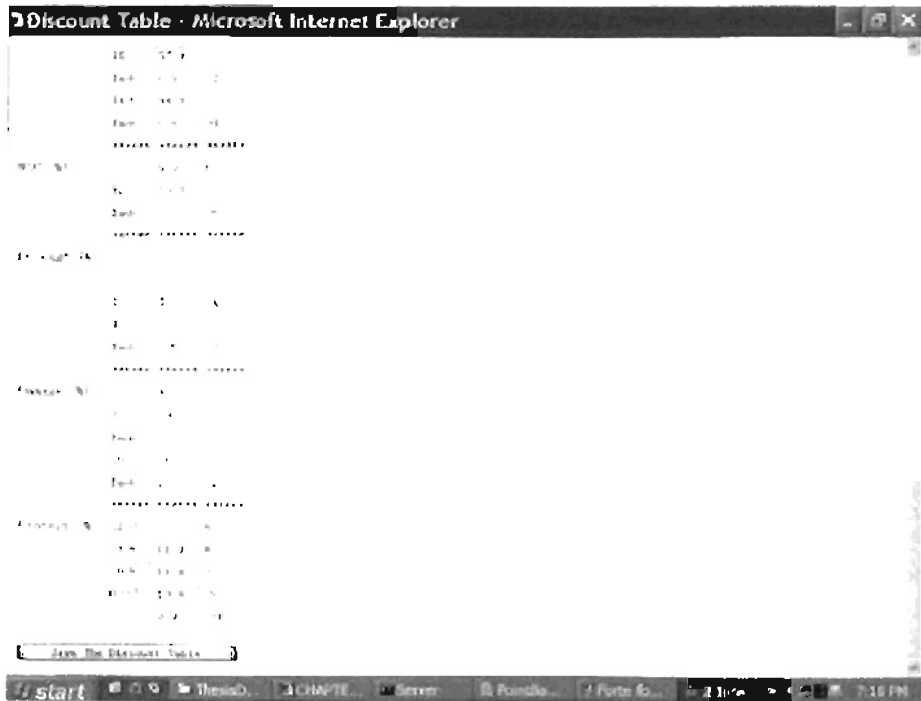


Figure 5.15 EditDisTable Pop-up Window

- **EditGradeTable** Pop-up Window (Figure 5.16)

The proposed security layer 7 “Role Based Access Control” (Chapter 3, section 3.3.2) is also embedded in the EditGradeTable pop-up window. If a

user plays a **role** (Chapter 2, section 2.4) as a System Administrator or a Web Application Designer, after he logs in, the server side gets his role from the back-end support database and sends the Grade Table pop-up window (Chapter 4) with an “Edit” button. By pressing the “Edit” button, the user can edit grade table in the EditGradeTable pop-up window.

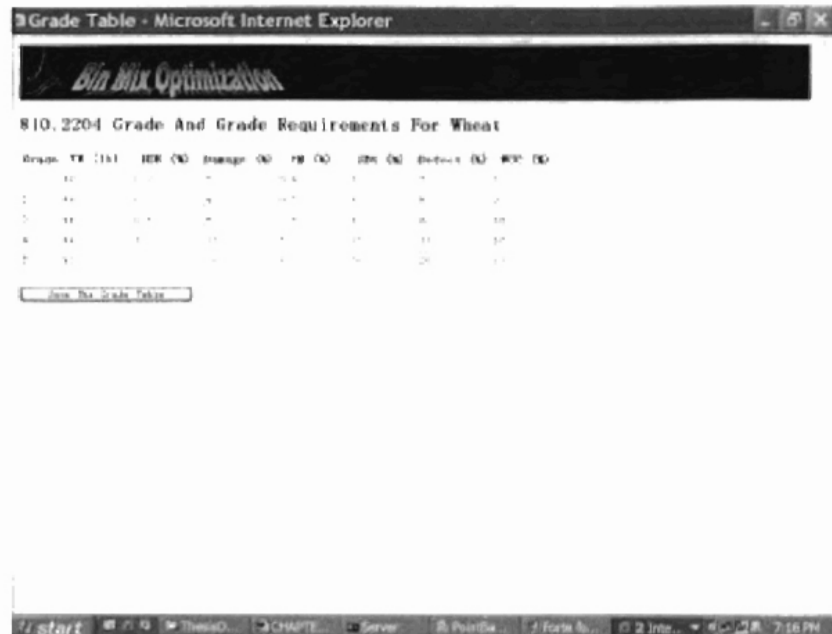


Figure 5.16 EditGradeTable Pop-up Window

5.3 Server Side Security Model Implementation

5.3.1 Defining Input Web Page Domain

Based on the architecture of the protected wheat bin mix optimization web application (Figure 5.1), we apply the “Input Web Page Domain Checking” technique

proposed in Chapter 3. We now define input web page domain for each web page of the protected wheat bin mix optimization web application as following (Table 5.1):

Table 5.1: Input Web Page Domain of Each Web Page

Web page	Input Web Page Domain
BinWebSite	{Null, Exit}
SetCookies	{BinWebSite}
TestCookies	{SetCookies}
CustRegist	{BinWebSite, RedoRegist, MustGive}
Error1	{TestCookies}
Error2	{TestCookies}
RegistDisplay	{CustRegist}
MustGive	{RegistDisplay}
RedoRegist	{RegistDisplay}
Login	{TestCookies, Error1, LoginError}
Check	{Login}
LoginError	{Check}
NoChance	{LoginError}
HomePage	{Check, ListBinInfo}
BinInformation	{HomePage}
ListBinInfo	{BinInformation}
ReadBinInfo	{BinInformation}
ListBinFromFile	{ReadBinInfo}

5.3.2 Implementation

In this section, we will discuss how to build the eleven security layers (Chapter 3) of the proposed server-side security model into wheat bin mix optimization web application.

Layer 1: Sanitizing Browser Inputs

As shown in Figure 5.1, The Sanitizing Browser Input layer is built into the RegistDisplay (Section 5.2) web page. When a user clicks the “Submit” button in the CustRegist page (Section 5.2) to submit registration information, in order to prevent the user using special characters such as ! and & or embedding malicious HTML tags in the Browser Inputs (Chapter 2, Section 2.9) to do cross-site scripting (Chapter 2, Section 2.9), the server side detects those malicious characters by comparing the Browser Inputs with a list of server-side defined valid characters. In the protected wheat bin mix optimization web application design, the list of server-side defined valid characters is characters a-z, A-Z and 0-9. If any of characters other than server-side defined valid characters exists, the server side sends the RedoRegist page (Section 5.2) to the user. The JSP code implementing “Sanitizing Browser Inputs” is listed in Appendix.

Layer 2: Cookies Support Detector

As shown in Figure 5.1, the Cookies Support Detector layer is built into both SetCookies (Section 5.2) page and TestCookies (Section 5.2) page. When a user clicks the “Sign In” button in BinWebSite page (Section 5.2), the server side sends the SetCookies page to the user and sets test cookies to the user’s browser. The JSP code to setting cookies to the client’s web browser is listed in Appendix. The SetCookies page

provides a “Cookies Support Detector” button to the user. By pressing this button, the user sends “Cookies Support Detector” request to the sever side, the server side then gets cookies it sets to the user’s browser and tests the cookies’ length. If no cookies exist (In Java programming, cookies are String data type objects since cookies are actually string of characters. Therefore, cookies do not exist means that the value of the String data type object is **null**.), indicating the user’s browser doesn’t support cookies or the user disables cookies, the server side sends the Error1 page (Section 5.2) to inform the user that it uses URL rewriting and session objects to maintain session status without cookies. If the cookies’ length is greater than zero, indicating the user’s browser supports cookies, the server side then tests the cookies’ consistency, which means that the server side compares the cookies fetched from the user’s browser with the cookies sets to the user’s browser. If the cookies fetched from the user’s browser exactly match the cookies the server side sets to the user’s web browser in the SetCookies page, the server side then sends the Login page (Section 5.2) to the user. Otherwise, the server side sends the Error2 page (Section 5.2) to the user. The JSP code testing cookies fetched from the client’s web browser is listed in Appendix.

Layer 3: Authentication

As shown in Figure 5.1, the authentication layer is built into the Login page (Section 5.2). We use username/password authentication to authenticate users. When a user clicks the “Submit” button in the Login page, the username and password he or she entered is sent to the server side. The server side then checks the username and password from the back-end support database. In this application, PointBase 4.2 within Forte for JAVA 4.0 package [Sun Microsystem 2002] is used to create the back-end support

database. If the correct username and password can be found from the database, the server side then sends the Check page (Section 5.2) to the user. By pressing the “Do Optimization” button in the Check page, then the user can perform wheat bin mix optimization. If a user’s username or password is incorrect, the server side sends the LoginError page (Section 5.2) to the user. By clicking the back arrow in the LoginError page, the user can redo login procedure. A user has at most three times to retry login. If a user can’t provide the correct username and password, the server side blocks the user’s access and sends the NoChance page to the user. The JSP code implementing the “username and password” authentication is listed in Appendix.

Layer 4: Setting Temporary Digital Signed Cookies

As shown in Figure 5.1, the Setting Temporary Digital Signed Cookies layer is built into the Check page (Section 5.2).

To set digital signed cookies to the client’s browser, the server side:

1. Uses Digital Signature Algorithm (Chapter 2) to generate a key pair—private key and public key.
2. Uses Secure Hash Algorithm (Chapter 2) to generate message digest for a user’s role information.
3. Uses the message digest and the private key generated in step 1 and 2 as input parameters to the Digital Signature Algorithm to generate the digital signature for the user’s role information and stores it into cookies.

To verify digital signed cookies, the server side:

4. Uses Secure Hash Algorithm to re-generate message digest for a user’s role information.

5. Retrieves the digital signature (digital signed role information) from the client's browser.
6. Uses retrieved digital signature from step 5. The digital signature algorithm inputs are the public key generated in step 1 and the message digest generated in step 4. Digital signature algorithm uses the public key to decrypt the digital signature to get a message digest and compare it with the message digest generated in step 4. The digital signature algorithm returns a Boolean value. If both message digests are the same, the digital signature algorithm returns true as comparison result. Otherwise, it returns false.

If the digital signature algorithm returns true, the server side processes the client's access. Otherwise, the server side denies the client's access. The procedure for signature generation and signature verification is shown in Figure 5.17 and Figure 5.18.

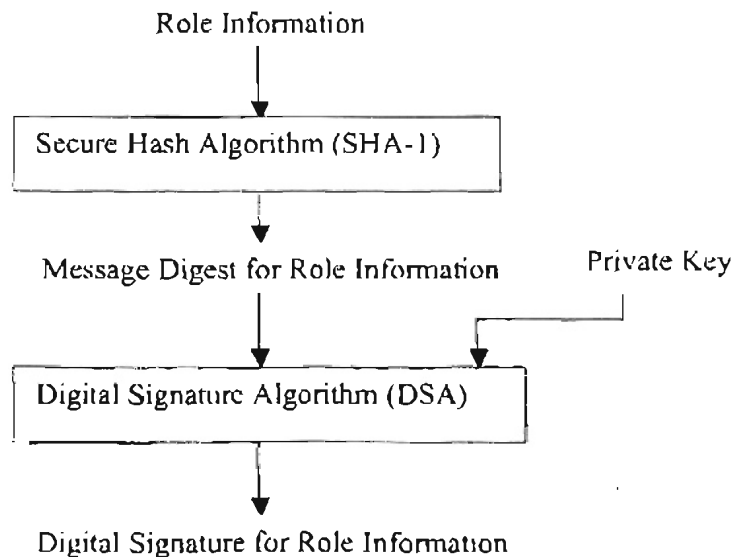


Figure 5.17 Digital Signature Generation

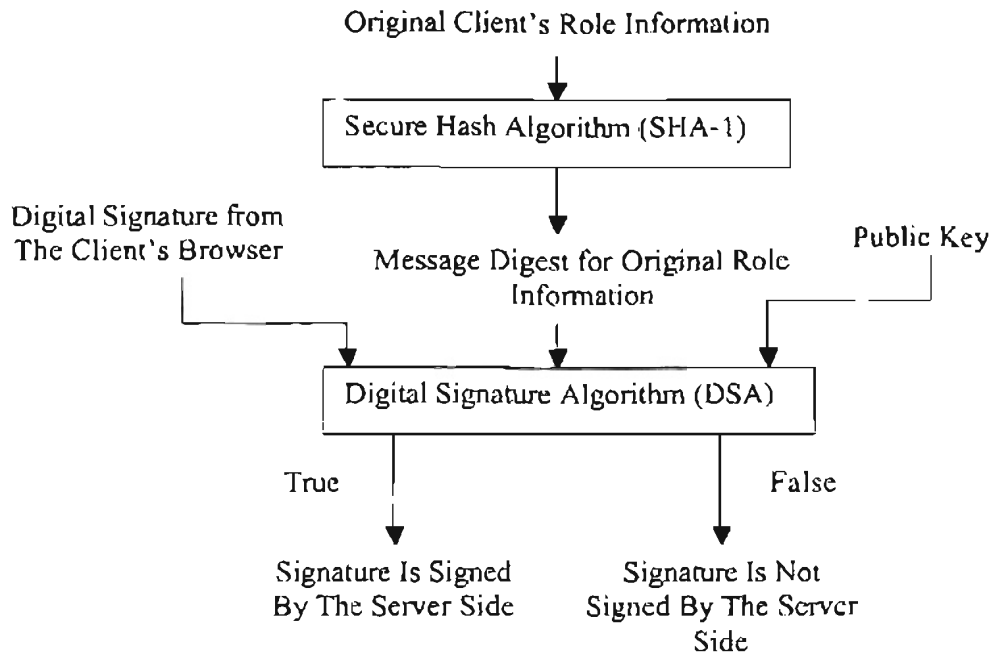


Figure 5.18 Digital Signature Verification

The JSP codes to set and get digital signed cookies to and from a user's browser and the JAVA class implementing digital signature is listed in Appendix.

Layer 5: URL Rewriting

As shown in Figure 5.1, the URL rewriting layer is built into the Check page (Section 5.2). After a user successfully logs in, a new session begins. If the client's browser doesn't support cookies, the server side rewrites the Universal Resource Location (URL) with session ID and stores the session ID into a session object.

An HTTP request is made up of the URL followed optionally by a query string containing pairs of parameters and values. For example, a HTTP request might be:

https://10.120.5.95:8443/Login.jsp;jsessionid=747795D4C986E03585D033730252C2B.

In this example, the server is *10.120.5.95*, the server resource is *Login.jsp*, and the session ID is *747795D4C986E03585D033730252C2B8*. Before URL rewriting, the URL was *https://10.120.5.95:8443/Login.jsp*. After URL rewriting, the URL becomes: *https://10.120.5.95:8443/Login.jsp;jsessionid=747795D4C986E03585D033730252C2B8*. Before the server side sends responses to the client's side, it compares the session ID passed by the request page with the session ID stored in the session object. If they match, the server side processes the client's request. Otherwise, the server side denies the client's request. The JSP code implementing URL rewriting is listed in Appendix.

Layer 6: Input Web Page Domain Checking

As shown in Figure 5.1, the Input Web Page Domain Checking layer is built into every web page. Each time a user accesses Wheat Bin Mix Optimization web site, to ensure the correct web application workflow, the server side needs to do input web page domain checking as discussed in Chapter 3. The input web page domain for each web page of wheat bin mix optimization web application is shown in Table 5.1. For example, the input web page domain for HomePage is {Check, ListBinInfo} as defined in Table 5.1. Before the server side sends the HomePage to the client's side, it checks whether a request comes from the Check page or the ListBinInfo page. If the request comes from one of those two pages, the server side sends the HomePage to the client's side. Otherwise, it denies the client's request. The JSP code implementing Input Web Page Domain Checking is listed in Appendix.

Layer 7: Role-Based Access Control

As shown in Figure 5.1, The Role-Based Access Control layer is built into the Discount (Chapter 4) and GradeTable (Chapter 4) pop-up windows. We assume that only System Administrators and Web Application Designers of the Wheat Bin Mix Optimization web site have the privilege to edit the grade and discount table. If a user sends the request for viewing the grade table or discount table to the server side, the server side first checks the user's role. If the user is a system administrator or web application designer of the Wheat Bin Mix Optimization web site, the server side sends the Discount and GradeTable pop-up window with an "Edit" button to the user. Otherwise, it just sends the Discount and GradTable pop-up window without an "Edit" button. The JSP code implementing Role-Based Access Control is listed in Appendix.

Layer 8: Hidden Form Field

The Hidden Form Field layer is built into each web page that passes parameters. Using Hidden Form Field, the parameters passed by web pages are not displayed by the client's web browser. For example, in the Login page (Section 5.2), after a user clicks the "Submit" button, the username and password are sent to the server side. Suppose the user's username is "Jack" and password is "12345". Without using hidden form field, the username and password information is displayed by a web browser that may look like the following: *https://10.120.5.95:8443/Login.jsp?username=Jack?password=12345*.

Obviously, passing parameters without using hidden form field in web pages is insecure. The JSP code implementing Hidden Form Field is listed in Appendix.

Layer 9: Deploying Secure Socket Layer As Needed

Since Secure Socket Layer (SSL) dramatically affects the speed at which users can access information, only sensitive web pages containing users' private information such as Login page and CustRegist page deploy SSL in the protected wheat bin mix optimization web application. In this application, we deploy Tomcat 4.0 [Apache Software Foundation 2003] as a web container. Tomcat 4.0 is a tool to make wheat bin mix optimization web application accessible from the web. Details about how to configure SSL supporting on Tomcat 4.0 are discussed in [Apache Software Foundation 2003].

Layer 10: Sensitive Web Page Caching Prevention

Sensitive web pages such as Login page, CustRegist page are prevented from being cached by the client's web browser. The JSP code implementing the sensitive web page caching prevention is listed in Appendix.

Layer 11--Killing cookies and Invalid Session

After a user clicks the "Exit" icon in each web page, the server side acknowledges that the current session is over and kills all cookies (sets cookies' lifetime to be zero) and sets the current session to be invalid. The JSP code implementing killing cookies and invalidating session is listed in Appendix.

CHAPTER 6

SUMMARY AND FUTURE WORK

6.1 Summary

This work proposed two techniques: Input Web Page Domain Checking and Cookies Support Detector. Input Web Page Domain Checking is a technique used to ensure a correct web application workflow, which means the server side sends each web page to the client side following a particular order. In Input Web Page Domain Checking, we first need to draw the architecture of a web application to reflect the application workflow. Secondly, we need to define the input web page domain for each web page based on the application workflow. Lastly, we do input web page domain checking according to the input web page domain for each web page.

Cookies Support Detector is a technique used to test whether the client's web browser supports cookies. In Cookies Support Detector technique, the server side sets cookies to the client's web browser then it gets the cookies from the client's web browser and tests the cookies' length. If the cookies length is greater than zero, the client's web browser supports cookies. Otherwise, the client's web browser doesn't support cookies or the client disables cookies.

This study also proposed a server side security model for web applications. The proposed security model consists of eleven security layers. The eleven security layers contains the proposed Input Web Page Domain Checking technique, the proposed Cookies Support Detector technique, and some existing techniques such as Sanitizing

Browser Inputs technique, Role-Based Access Control technique, and Setting Temporary Digital Signed Cookies technique. The Setting Temporary Digital Signed Cookies technique is modified from the "Role-Based Access Control Using Secure Cookies" technique proposed in [Park 2001]. In the modified Setting Temporary Digital Signed cookies technique, cookies properties are changed from persistent cookies to temporary cookies. Temporary cookies have two advantages. First, it is only stored in the client's web browser's memory and is killed after a session is over so it doesn't waste the storage of the client's hard disk. Second, it is more secure than persistent cookies since a user cannot retrieve the contents of a web page after a session is over.

To demonstrate the proposed sever side security model, a Wheat Bin Mix Optimization web application is developed. The server side security model is built into the web application.

6.2 Future Work

The proposed server-side security model can be built into any web applications to protect sensitive information because this model sets eleven security layers between the client's requests and the server side's responses. The proposed security model combines all advantages of currently most widely used secure measures with two proposed new techniques. However, since there are always new unknown types of attack, there is no guarantee to establish a "complete secure system". Protectors will propose and design more secure systems. At the same time, attacks will continue their efforts for breaking existing secure system. The proposed sever-side security model needs to improve and modify to face the future security challenge of web applications.

BIBLIOGRAPHY

- [Apache Software Foundation 2003] "SSL Configuration How To." [On-line], Available: <http://jakarta.apache.org/tomcat/tomcat-4.0-doc/ssl-howto.html>, Apache Software Foundation (Access Date: January 5, 2003)
- [Ayers 1999] Ayers, D., et.al., "Professional Java Server Programming.", *Wrox Press Ltd.* (1999)
- [Berry 1994] Berry, B. and Taila Booth, "Informix-Online Dynamic Server Administrator's Guide." *Version 7.1. INFORMIX Software, Inc.* (1994)
- [Cook 2000] Cook, J., Robert Harbus and Tetsuya Shirai, "DB2 Universal Database." *V6.1, 2nd Edition, Prentice Hall* (2000)
- [Duffey 2001] Duffey, K., et.al., "Profession JSP Site Design.", *Wrox Press Ltd.* (2001)
- [Federal Grain Inspection Service 2002] "Official United States Standards for Grain." *Federal Grain Inspection Service* (2002)
- [Ferraiolo 2003] Ferraiolo, D. F., D. R. Kuhn, and R. Chandramouli, "Role Based Access Control" *ISBN: 1-58053-370-1, Artech House Publisher* (2003)
- [Garfinkel 1997] Garfinkel, S. and G. Spafford, "Web Security and Commerce." *O'Reilly and Associates, Sebastopol, CA.* (1997)
- [George 1997] George, K. and Kevin Loney, "Oracle 8: The Complete Reference." *Osborne McGraw-Hill* (1997)
- [Goldberg 2002] Goldberg, I. K. "Glossary Of Information Warfare Terms." [On-line], Available: <http://www.psycom.net/iwar.2.html>. (Access Date: October 10, 2002)
- [Joshi 2001] Joshi, J. B. D., W. G. Aref, A. Ghafoor, and E. H. Spafford, "Security Models for Web-based Applications." *Communication of the ACM, Vol. 44, No. 2, pp. 38-44* (2001)
- [Lewis 2000] Lewis, R. M., V. Torczon and M. W. Trosset, "Direct Search Methods: Then and Now." *Journal of Computational and Applied Mathematics, Vol. 124, pp. 191-207* (2000)
- [Loshin 1999] Loshin, P., "Big Book of Ipsec RFCs Internet Security Architecture." *Morgan Kaufmann Publishers* (1999)

[Naor 1990] Naor, M., and M. Yung, "Public-Key Cryptosystems Provable Secure Against Chosen Ciphertext Attacks." *ACM Press, New York*, pp. 427-437 (1990)

[Neuman 1994] Neuman, B. C. and Theodore TS'O, "Kerberos: an Authentication Service for Computer Networks." *IEEE Communications, Vol.32, Number 9*, pp.33-38. (1994)

[Oracle Company 1999] "Database Security in Oracle8i." *Oracle Technical White Paper [On-line]*. Available: <http://otn.oracle.com/deploy/security/nissc00.htm> (Access Date: Oct. 15, 2002)

[Park 2001] Park, J.S., R. Sandhu, and G. J. Ahn, "Role Based Access Control on The Web." *Communications of ACM, Vol. 4, Issue 1*, pp.37-71. (2001)

[Peavey Company 2000] "Grade Discount in Cents Per Bushel." *Peavey Company, 5301 West Channel Road, Catoosa, Oklahoma 74015*. (2000)

[Pettit 2001] Pettit, S., "Anatomy of a Web Application: Security Considerations." *[On-line]* Available: http://www.cgisecurity.com/lib/Web_Server.pdf. (Access Date: October 15, 2002)

[Rankle 1997] Rankl, W. and Wolfgang Effing, "Smart Card Handbook." *John Wiley & Sons Ltd.* (1997)

[Rosenbeery 1992] Rosenbeery, K., David Kenney and Gerry Fisher, "Understanding DCE." *O'Reilly & Associates, Inc.* (1992)

[Schwartz 2001] Schwartz, R., "Using Field Encryption in Applications." *[On-line]* Available: [http://www-10.lotus.com/ldd/today.nsf/8abd147cf55a7fd385256658007aacf1/24d3f7b03bcaf0c388256abb00730519/\\$FILE/encrypt.pdf](http://www-10.lotus.com/ldd/today.nsf/8abd147cf55a7fd385256658007aacf1/24d3f7b03bcaf0c388256abb00730519/$FILE/encrypt.pdf) Iris, Association Inc. (Access Date: September 30, 2002)

[Stillerman 1999] Stillerman, M., and C. Marceall, "Intrusion Detection for Distributed Applications." *Communications of the ACM, Vol. 42, Issue 7*, pp.62-69. (1999)

[Sun Microsystem 2002] "Forte for Java 4, Community Edition Tutorial." *[On-line]* Available: <http://forte.sun.com/fff/documentation/ffjcetut.pdf>. Sun Microsystems, Inc. (Access Date: September 30, 2002)

[Xtream 2002] Xtream, "Internet Security." *[On-line]* Available: <http://xtream.online.fr/project/security.html>. (Access Date: September 29, 2002)

[Yen 1995] Yen, S.M., and C. S. Lai, "Improved Digital Signature Algorithm" *IEEE Transaction On Computers, Vol. 44, No. 5* (1995)

APPENDIX A

JSP CODES AND JAVA PROGRAMS FOR SECURITY LAYERS

SANITIZING BROWSER INPUTS

*/*A Java class named "ValidateInput used to check whether a user's input contain invalid characters.*/*

```
public class ValidateInput {
    /** Creates a new instance of ValidateInput */
    public ValidateInput() {}
    public int checkValidInput(String s){
        int flag=0;
        for(int i=0; i<s.length(); i++){
            //the server-side defined valid characters are a-z, A-Z and 0-9;
            char inChar=s.charAt(i);
            if (((inChar > a)&&(inChar < z)) || ((inChar > A)&&(inChar < Z)) || ((inChar >
0)&&(inChar < 9)))
                flag=1;
            else {
                flag=-1;
                break;
            }
        }
        return flag;
    }
}
```

/ A JSP code in RegistDisplay JSP page used to call ValidateInput class. If a user's registration information contain any invalid character, that is, if the return value of ValidateInput class is "-1", the RegistDisplay page just forward RedoRegist page to the user. Otherwise, it displays the user's registration information.*/*

```
<%@page import="BinCustomer.*" %>
```

```
<a href="https://10.120.5.95:8443/RegistDisplay.jsp"></a>
```

```
<%@include file="GetCustId.jsp"%>
```



```

<jsp:useBean id="custBean" class="BinCustomer.Customer" scope="session"/>
<jsp:useBean id="userBean" class="BinCustomer.userTable" scope="application"/>
<jsp:useBean id="validate" class="BinCustomer.ValidateInput" scope="session"/>

<jsp:setProperty name="userBean" property="*" />
<jsp:setProperty name="validate" property="*" />
<jsp:setProperty name="custBean" property="*" />

<html>
<head>

<title>Regist Display Page</title></head>
<body >

<TABLE class="header" width="100%" cellspacing="0" cellpadding="3" border="0"
bgcolor="#000000">
  <TR>
    <TD></TD>
    <TD align="center" width="35">
      <TD align="center" width="35">
        <a href="http://10.120.5.95:8081/ExitWebSite.jsp">
          <IMG src="exit_16.gif" height="16" width="16" border="0"><br>
          <Font size="2">LogOut</Font></A></TD></TR>
</TABLE>

<% if( (validate.checkValidInput(request.getParameter("userName"))== -1)||
      (validate.checkValidInput(request.getParameter("myPassword"))== -1)||
      (validate.checkValidInput(request.getParameter("firstName"))== -1)||
      (validate.checkValidInput(request.getParameter("middleName"))== -1)||
      (validate.checkValidInput(request.getParameter("lastName"))== -1)||
      (validate.checkValidInput(request.getParameter("streetNumber"))== -1)||
      (validate.checkValidInput(request.getParameter("street"))== -1)||
      (validate.checkValidInput(request.getParameter("apt"))== -1)||
      (validate.checkValidInput(request.getParameter("city"))== -1)||
      (validate.checkValidInput(request.getParameter("state"))== -1)||
      (validate.checkValidInput(request.getParameter("zip"))== -1)||
      (validate.checkValidInput(request.getParameter("telephoneNumber"))== -1)) {

%>
<jsp:forward page="RedoRegist.jsp">
<jsp:param name="RegistDisplay" value="yes"/>
</jsp:forward>
<%
} //end if flag== -1

```

```

custBean=new Customer();
custBean.setMyPassword(request.getParameter("myPassword"));
custBean.setUserName(request.getParameter("userName"));

//String temp=" ";
custBean.setFirstName(request.getParameter("firstName"));
//if(request.getParameter("middleName")==null)
// temp=" ";
custBean.setMiddleName(request.getParameter("middleName"));
custBean.setLastName(request.getParameter("lastName"));
custBean.setStreetNumber(request.getParameter("streetNumber"));
custBean.setStreet(request.getParameter("street"));
custBean.setApt(request.getParameter("apt"));
custBean.setCity(request.getParameter("city"));
custBean.setState(request.getParameter("state"));
custBean.setZip(request.getParameter("zip"));
custBean.setTelephoneNumber(request.getParameter("telephoneNumber"));

%>

<% custBean.recordCustomer();
%>

<H2>Welcome, <jsp:getProperty name="custBean" property="userName" /> </H2>

<p><i>We have recorded the following data in your profile:</i></p>

<table>
  <tr>
    <td class="registerTD">First Name:</td>
    <td><jsp:getProperty name="custBean" property="firstName" /></td>
  </tr>
  <tr>
    <td class="registerTD">Middle Name:</td>
    <td><jsp:getProperty name="custBean" property="middleName" /></td>
  </tr>
  <tr>
    <td class="registerTD">Last Name:</td>
    <td><jsp:getProperty name="custBean" property="lastName" /></td>
  </tr>
  <tr>
    <td class="registerTD">Street Number:</td>
    <td><jsp:getProperty name="custBean" property="streetNumber" /></td>
  </tr>

```

```

<tr>
  <td class="registerTD">Street:</td>
  <td><jsp:getProperty name="custBean" property="street" /></td>
</tr>
<tr>
  <td class="registerTD">Apartment:</td>
  <td><jsp:getProperty name="custBean" property="apt" /></td>
</tr>
<tr>
  <td class="registerTD">City:</td>
  <td><jsp:getProperty name="custBean" property="city" /></td>
</tr>
<tr>
  <td class="registerTD">State:</td>
  <td><jsp:getProperty name="custBean" property="state" /></td>
</tr>
<tr>
  <td class="registerTD">Zip:</td>
  <td><jsp:getProperty name="custBean" property="zip" /></td>
</tr>
<tr>
  <td class="registerTD">Telephone:</td>
  <td><jsp:getProperty name="custBean" property="telephoneNumber" /></td>
</tr>
</table>
</p>
</p>
</body>
</html>

```

COOKIE SUPPORT DETECTOR

/ A JSP page used to set cookies to the client's web browser */*

```

<%@page contentType="text/html"%>
<%@page import="javax.servlet.http.Cookie"%>

<a href="https://10.120.5.95:8443/setCookies.jsp"></a>

<body>

<TABLE class="header" width="100%" cellspacing="0" cellpadding="3" border="0"
bgcolor="#000000">
  <TR>
    <TD></TD>

```

```
</TR>
</TABLE>
```

```
<%
Cookie newCookie=new Cookie("binCookie", "wheatBin");
response.addCookie(newCookie);
%>
```

```
<form method=post action="https://10.120.5.95:8443/testCookies.jsp">
<input type=hidden name=setCookies value="yes">
<input type=submit name=operation value="Cookies Support Detector">
</form>
```

```
</body>
</html>
```

/ A JSP page used to test whether the client's browser supports cookies*/*

```
<%@page contentType="text/html"%>
<%@page import="javax.servlet.http.Cookie"%>
<html>
<head><title>JSP Page</title></head>
<body>
```

```
<%! Cookie[] cookies;%>
<%! String cookieVal="";%>
```

```
<%
    if(request.getCookies()==null){// The user's browser may not support cookies or
cookies are disabled
%>
<jsp:forward page="Error1.jsp">
<jsp:param name="testCookies" value="yes"/>
</jsp:forward>
<%
}

```

```
cookies=request.getCookies();
if(cookies!=null){
    for(int i=0; i<cookies.length; i++){
        if(cookies[i].getName().equals("binCookie")){
            cookieVal=cookies[i].getValue();
            break;
        }
    }
}
```

```

    }

    if (cookieVal!=null){..The user's browser support cookies but we can not find the
    cookies we set
    %>
    <jsp:forward page= "Error2.jsp">
    <jsp:param name="testCookies" value="yes"/>
    </jsp:forward>
    <%
    }

```

```

if(cookieVal.equals("wheatBin")){//The user's browser support cookies, We set digital
signature cookies to the user's web browser
%>
<TABLE class="header" width="100%" cellspacing="0" cellpadding "3" border="0"
bgcolor="#000000">
  <TR>
    <TD></TD>
  </TR>
</TABLE>

</p>
<h3> Your browser support cookies, We will use cookies to maintain session
information.</H3>
</p>
<form method=post action="https://10.120.5.95:8443/Login.jsp">
<input type=hidden name=testCookies value="yes">
<input type=submit value="I Agree">
</FORM>
<%
}
%>

</body>
</html>

```

AUTHENTICATION

/ Login JSP page which uses username/password to authenticate users*/*

```

<%@page contentType="text/html"%>
<%@page session="true"%>
<a href="https://10.120.5.95:8443/Login.jsp"></a>

<%-- Login page can not be cached by the user's web browser! --%>

```

```

<%
response.setHeader("Cache-Control","no-cache"); //HTTP 1.1
response.setHeader("Pragma","no-cache"); //HTTP 1.0
response.setDateHeader ("Expires", 0); //prevents caching at the proxy server
%>
<%-- Do input page domain checking --%>
<%! int inputPageFlag;%>
<%
    inputPageFlag=0;
    if(request.getParameter("testCookies")!=null){
        if(request.getParameter("testCookies").equals("yes")){
            inputPageFlag=1;//support cookies
        }
    }
    if(request.getParameter("Error1")!=null){
        if(request.getParameter("Error1").equals("yes")){
            inputPageFlag=2;//using session object
        }
    }
    if(request.getParameter("LoginError")!=null){
        if(request.getParameter("LoginError").equals("yes")){
            inputPageFlag=3;//redo login
        }
    }
    if(inputPageFlag==0){
%>
-jsp:forward page="pageNotFound.jsp"
<%
}
%>
<%-- in case malicious user user back button and refresh button to get the sensitive page -
-%>
<% Cookie[] allcookies request.getCookies();
String realTest=null;
if(allcookies!=null){
    for(int i=0; i<allcookies.length;+ i){

        if(allcookies[i].getName().equals("binCookie")){
            realTest=allcookies[i].getValue(),
            break;
        }
    }
}

if(realTest !=null){
%>

```

```

<jsp:forward page="pageNotFound.jsp"/>
<%
}
}
else{//cookies not supported
    if(session.getValue("sessionOk").equals("OK")==false){
%>
<jsp:forward page="pageNotFound.jsp"/>
<%
}
} //end else cookies not supported
%>
<html>
<head><title>Log In</title></head>
<body>
<TABLE class="header" width="100%" cellspacing="0" cellpadding="3" border="0"
bgcolor="#000000">
    <TR>
        <TD></TD>
        <TD align="center" width="35">
            <TD align="center" width="35">
                <a href="http://10.120.5.95:8081/ExitWebSite.jsp">
                    <IMG src="exit_16.gif" height="16" width="16" border="0"><br>
                    <Font size="2">LogOut</Font></A></TD></TR>
</TABLE>
</p>
<% if(inputPageFlag==1){
%>
<form name="login" method="POST" action="https://10.120.5.95:8443/Check.jsp">
<%
}
if(inputPageFlag==2){
    String url;
    url=response.encodeUrl("Check.jsp");
%>
<form name="login" method="POST" action="<%= url %>">
<%
}
%>
<table>
    <tr>
        <td class="loginMenu">User ID&nbsp;&nbsp;&nbsp;</td>
        <td>
            <input name="user" type="text" length="8" maxlength="8" />
        </td>
    </tr>

```

```

        <tr>
            <td class="loginMenu">Password&nbsp;</td>
            <td>
                <input name="password" type="password" length="8" maxlength="8" />
            </td>
        </tr>
    </table>
<p>
<%! String temp; %>
<% if(request.getParameter("timeCount") == null)
    temp="three";
    else
        temp=request.getParameter("timeCount");
%>

<% if (inputPageFlag == 1){
%>
<input type="hidden" name="cManager" value="yes">
<%
}
    if (inputPageFlag==2){
        session.putValue("loginOk","OK");//used to maintain session consistency
%>
<input type="hidden" name="sManager" value="yes">
<%
}
//add from here

%>
<input type="hidden" name="Login" value="yes">
<input type="hidden" name="passValue" value="<%= temp%>"/>
<input type="submit" value="Login" class="btn" />
<input type="reset" value="Reset" class="btn" />
</p>
</form>

<% session.putValue("sessionOk","No");%>
</body>
</html>

```


SETTING TEMPORARY DIGITAL SIGNATURE COOKIES

/* The following is the Java program used to generate and verify digital signature*/

```
package DigitalSignature;
import java.io.*;
import java.security.*;
import java.math.BigInteger;
import java.security.interfaces.DSAParams;
import java.security.interfaces.DSAPrivateKey;
import java.security.interfaces.DSAPublicKey;
import java.security.Signature.*;
import java.security.spec.*;
import java.security.interfaces.DSAParams;
import java.security.KeyStore;

public class digitalSignature {

    public DSAPublicKey idPubKey;
    public DSAPublicKey tryKey;
    public DSAPublicKey rolePubKey;
    public int idBackUpLen;
    public int roleBackUpLen;
    private BigInteger yy;
    private BigInteger pp;
    private BigInteger qq;
    private BigInteger gg;
    public String strY;
    public String strP;
    public String strQ;
    public String strG;

    public digitalSignature(){}
    public byte[] GenSign(String str, String flag){

        try{

            // Generate a 1024-bit Digital Signature Algorithm (DSA) key pair
            KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA");
            keyGen.initialize(1024);
            KeyPair keypair = keyGen.genKeyPair();
            DSAPrivateKey privateKey = (DSAPrivateKey)keypair.getPrivate();
            DSAPublicKey publicKey = (DSAPublicKey)keypair.getPublic();

            //*****
            tryKey =publicKey;
```

```

// Get p, q, g; they are the same for both private and public keys
DSAParams dsaParams = privateKey.getParams();
pp = dsaParams.getP();
qq = dsaParams.getQ();
gg = dsaParams.getG();
// Get the public key's Y
yy = publicKey.getY();

strP=pp.toString();
strQ=qq.toString();
strG=gg.toString();
strY=yy.toString();

// Generate a digital signature using private key
//Get a signature Object for generating signature using the DSA algorithm.
//specify SHA1 message digest algorithm used by DSA algorithm.
Signature mySign = Signature.getInstance("SHA1withDSA", "SUN");
mySign.initSign(privateKey);

//Now, provide the signature object mySign the cookie value infor to be signed.
mySign.update(inputStrToByte(str));

//Now, generate the signature
byte[] finSign = mySign.sign();

if(flag.equals("id"))
    idBackUpLen=finSign.length;//set the back up length for convert digital signed
id back to byte[]
if(flag.equals("role"))
    roleBackUpLen=finSign.length;//set the back up length for convert digital
signed role back to byte[]
// return temp;
return finSign;
} catch (SignatureException e) {
} catch (InvalidKeyException e) {
} catch (NoSuchAlgorithmException e) {
} catch (NoSuchProviderException e){
}
return null;
}

public boolean verifySign(byte[] sign, String str, DSAPublicKey k){
// Verifies the signature for the given string using the public key.
try {

```

```

Signature sig = Signature.getInstance("SHA1withDSA", "SUN");

//if(flag=="id")
    sig.initVerify(k);
//else
    // sig.initVerify(rolePubKey);

sig.update(inputStrToByte(str));

//return sig.verify(strToByte(sign));
return sig.verify(sign);
} catch (SignatureException e) {
} catch (InvalidKeyException e) {
} catch (NoSuchAlgorithmException e) {
} catch (NoSuchProviderException e){
}
return false;
}

public byte[] inputStrToByte(String s){

int temp=0;

// convert a String into a byte array
byte buf[] = new byte[s.length()+1];
for (int i=0; i<s.length(); i++) {
    temp =(int)(s.charAt(i));
    buf[i]=(byte)(temp);
}
return buf;
}

public String byteArrToStr(byte[] bi){

Byte[] myByte=new Byte[bi.length];
String s="";

for(int j=0;j<bi.length;j++){
    myByte[j]=new Byte(bi[j]);
    s=s+myByte[j].toString()+" ";
}
return s;
}

public byte[] strToByteArr(String s, int len){

```

```

byte[] newByte;
int tIndex=0;
int index;

newByte=new byte[len];

index=s.indexOf(" ");
for(int k=0; k<len; k++){
    String temp=s.substring(tIndex,index);
    newByte[k]=Byte.parseByte(temp);
    tIndex=index+1;
    index=s.indexOf(" ",tIndex);
    if(index==-1)
        break;
}

return newByte;
}

public void rePubKey(String yy,String pp,String qq, String gg, String flag){
    try{
        BigInteger y=new BigInteger(yy);
        BigInteger p=new BigInteger(pp);
        BigInteger q=new BigInteger(qq);
        BigInteger g=new BigInteger(gg);

        // Create the DSA key factory
        KeyFactory keyFactory = KeyFactory.getInstance("DSA");

        // reCreate the DSA public key
        DSAPublicKeySpec publicKeySpec = new DSAPublicKeySpec(y, p, q, g);
        DSAPublicKey publicKey = (DSAPublicKey)keyFactory.generatePublic
        (publicKeySpec);
        if(flag == "id")
            idPubKey=publicKey;
        else
            rolePubKey=publicKey;
    }catch(NoSuchAlgorithmException e){
    }catch(InvalidKeySpecException e){
    }
}
}

```

/* The following JSP code is used to call digitalSignature class and set digital signed cookies to the client's browser. */

```

<!-- Delete all existing cookies and add new cookies with digital signital-->
<%
String name="";
//check if there are any cookies and delete all existing cookies!
if(request.getCookies()!=null){
    Cookie[] cookies=request.getCookies();
    for(int i=0;i<cookies.length;i++){
        name=cookies[i].getName(); //get each cookie name
        //kill each existing cookie
        Cookie killCookie = new Cookie(name, null);
        killCookie.setMaxAge(0);
        response.addCookie(killCookie);
    }
}

//set new cookie.
user=request.getParameter("user");
//put role information into session object
roleStr=getRole.roleInfo(user);

//digital signature the id cookie value.
byteArr=digi.GenSign(roleStr,"id");
syy=digi.strY;
spp=digi.strP;
sqq=digi.strQ;
sgg=digi.strG;
len=digi.idBackUpLen;

%>

<%

String cos =digi.byteArrToStr(byteArr);
Cookie nameCookie =new Cookie("roleName", cos);
nameCookie.setMaxAge(-1);
response.addCookie(nameCookie);
%>
<form method=post action="http://10.120.5.95:8081/HomePage.jsp">
<input type=hidden name=cManager value="yes">
<input type=hidden name=syy value="<%=syy%>"/>
<input type=hidden name=spp value="<%=spp%>"/>
<input type=hidden name=sqq value="<%=sqq%>"/>
<input type=hidden name=sgg value="<%=sgg%>"/>
<input type=hidden name=len value="<%=len%>"/>
<input type=hidden name=roleStr value="<%= roleStr%>">

```

·:·%

URL REWRITING

/* The following JSP code is used to do URL rewriting*/

```
<%! String url;·%>
<%
    user=request.getParameter("user");
    //put role information into session object
    roleStr=getRole.roleInfo(user);
    session.putValue("roleStr",roleStr);
    url =response.encodeURL("HomePage.jsp");
%>
<%-- --Using URL rewriting to maintain session status-----%>
<form method=post action="<%= url %>">
<input type=hidden name=sManager value="yes">
<input type=hidden name=sId value="<%= session.getId()%>">
<%
    }//end sManager is yes
} //end sManager is not null
%>
<% session.putValue("checkOk", "OK");%>
<input type=hidden name=check value="yes">
<input type=hidden name=user value="<% request.getParameter("user")%>"·
<input type=hidden name=password value="<%=request.GetParameter
("password")%>" />
<input type=submit value="Do Optimization">
</form>
```

INPUT WEB PAGE DOMAIN CHECKING

/* For example, the input web page domain for Login web page is {TestCookies, ErrorI, LoginError}. The following JSP code in Login page is used to do input web page domain checking. */

```
<%-- Do input page domain checking --%>
<%! int inputPageFlag;%>
<%
    inputPageFlag=0;
    if(request.getParameter("testCookies")!=null){
        if(request.getParameter("testCookies").equals("yes")){
            inputPageFlag=1: /support cookies
        }
    }
%>
```

```

    }
}
if(request.getParameter("Error1")!=null){
    if(request.getParameter("Error1").equals("yes")){
        inputPageFlag=2;//using session object
    }
}
if(request.getParameter("LoginError")!=null){
    if(request.getParameter("LoginError").equals("yes")){
        inputPageFlag=3;//redo login
    }
}
if(inputPageFlag= 0){
%>
<jsp:forward page="pageNotFound.jsp"/>
<%
}
%>

```

ROLE-BASED ACCESS CONTROL

/* For example: if a user clicks "Grade Table" or "Discount Table" button in the ListBinInfo page, the server side sends the grade table pop-up window or discount pop-up window depends on the user's role. */

```

//if the client's browser doesn't support cookies
<% if(flag.equals("session")){
    if(session.getValue("roleStr").equals("Customer")){
%>
<FORM method="POST" action="javascript:PopUp
('http://10.120.5.95:8081/GradeTable.jsp')" >
<p><input type=image src="help_16.gif"> Grade Table</p>
</FORM>
<FORM method="POST" action="javascript:PopUp
('http://10.120.5.95:8081/Discount.jsp')" >
<p><input type=image src="help_16.gif"> Discount Table</p>
</FORM>
<%
}
    if(session.getValue("roleStr").equals("Admin")){
%>
<FORM method="POST" action="javascript:PopUp
('http://10.120.5.95:8081/EditGradeTable.jsp')" >
<p><input type=image src="help_16.gif"> Grade Table</p>
</FORM>

```

```

<FORM method="POST" action="javascript:PopUp
('http://10.120.5.95:8081/EditDisTable.jsp')" >
<p><input type="image" src="help_16.gif"> Discount Table</p>
</FORM>
<%
}
}
%>

//if the client's browser supports cookies
<% if(flag.equals("cookies")){
    if(request.getParameter("roleStr").equals("Customer")){
%>
<FORM method="POST" action="javascript:PopUp
('http://10.120.5.95:8081/GradeTable.jsp')" >
<p><input type="image" src="help_16.gif"> Grade Table</p>
</FORM>
<FORM method="POST" action="javascript:PopUp
('http://10.120.5.95:8081/Discount.jsp')" >
<p><input type="image" src="help_16.gif"> Discount Table</p>
</FORM>
<%
}
    if(request.getParameter("roleStr").equals("Admin")){
%>
<FORM method="POST" action="javascript:PopUp
('http://10.120.5.95:8081/EditGradeTable.jsp')" >
<p><input type="image" src="help_16.gif"> Grade Table</p>
</FORM>
<FORM method="POST" action="javascript:PopUp('http://10.120.5.95:8081/EditDisTable.jsp')" >
<p><input type="image" src="help_16.gif"> Discount Table</p>
</FORM>
<%
}
}
%>

```

HIDDEN FORM FILED

/* The following JSP code in Login JSP page uses hidden form field to send a user's username and password to the check page. */

```

<form name="login" method="POST" action="https://10.120.5.95:8443/Check.jsp">
<input type="hidden" name="Login" value="yes">

```



```

<input type= hidden name= passValue value="<%-- temp%>"/>
<input type="submit" value ="Login" class="btn" >
<input type="reset" value="Reset" class "btn" />
</p>
</form>

```

SENSITIVE WEB PAGE CACHING PROTECTION

/* The following JSP code in ListBinInfo JSP page is used to prevent this page from being cached by the client's web browser. */

```

<%-- ListBinInfo can not be cached by the user's web browser! --%>
<%
response.setHeader("Cache-Control","no-cache"); //HTTP 1.1
response.setHeader("Pragma","no-cache"); //HTTP 1.0
response.setDateHeader ("Expires", 0); //prevents caching at the proxy server
%>

```

KILLING COOKIES AND INVALIDATING SESSION

/* The following JSP code in Exit JSP page is used to kill cookies and invalid session when a session is over. */

```

<%
//kill all existing cookies
String name="";
//check if there are any cookies and delete all existing cookies!
if(request.getCookies()!=null){
    Cookie[] cookies=request.getCookies();
    for(int i =0;i<cookies.length;i+ ){
        name=cookies[i].getName();//get each cookie name
        //kill each existing cookie
        Cookie killCookie = new Cookie(name, null);
        killCookie.setMaxAge(0);
        response.addCookie(killCookie);
    }
}
//kill role session object
session.putValue("roleStr", "empty");
session.putValue("sessionOk", "No");
session.invalidate(); //invalid session
%>

```

VITA

2

Jing Ding

Candidate for the Degree of
Master of Science

Thesis: A SERVER-SIDE SECURITY MODEL FOR WEB APPLICATIONS

Major Field: Computer Science

Biographical:

Personal Data: Born in Hefei, China, On Oct. 31, 1974.

Education: Received Bachelor of Science degree in Chemical Engineering from Hefei United University, Hefei, China in July 1995. Completed the requirements for the Master of Science degree in Computer Science at Oklahoma State University in December 2003.

Experience: Employed by Hefei Phoenix Medical Equipment Inc., Hefei, China, as a technical support engineer, 1995-1998.

Professional Membership: Phi Kappa Phi Honor Society Member.