An Investigation into Server-side Static and Dynamic Web Content Survivability Using a Web Content Verification and Recovery (WCVR) System

Shadi Aljawarneh

School of Computing, Engineering & Information Sciences

Northumbria University

A thesis submitted for the degree of

Doctor of Philosophy (PhD)

26th of June 2008

An Investigation into Server-side Static and Dynamic Web Content Survivability Using a Web Content Verification and Recovery (WCVR) System

Declaration

I declare that the work contained in this thesis has not been submitted for any other award and that it is all my own work.

Name:

Signature:

Date:

I would first like to thank my GOD who give me the patience and ability to write this thesis. As well as, I would like to dedicate this thesis to my loving parents (Abu loui, and Om Loui) and my family (Lina, and Ayshah), who offered me a great love and support throughout the course of this thesis.

Acknowledgements

Firstly, I would like to sincerely thank my supervisors Dr. Christopher Laing, Dr. Paul Vickers, and Prof. Maia Anglova for their constant guidance, great friendship and for their unique and charming way of support and assistance.

Much of the work in this thesis would have been impossible without the generosity of Dr. Christopher Laing, and Dr. Paul Vickers. Dr. Laing helped me identify for all elements in my research work from the first stage to the final draft. In retrospect, I feel that I have been very lucky to have someone who has such good insight in web security research, and gives me the most crucial encouragement at the most difficult times. Dr. Vickers learned me many things in the life besides his academic advice. He has a far-reaching insight in the academic research. I am grateful to Prof. Maia Anglova for her support and assistance in data modelling.

A special thanks as well to the postgraduate director Dr. Nick Rossiter who I am deeply grateful for the opportunity he gave to me to be a member in family of Northumbria University. I would also like to thank K, R, and J for running the experimental test. I acknowledge Al-Asra University for their financial support. I thank all my friends for their encouragement and friendship.

I would also like to take this opportunity to express my gratitude to my family members (my parents, my wife, my daughter, my brother Loui, my uncle Dr. Ibrahim, and all brothers and sisters) for their love, unfailing encouragement and support. Finally, I would be remiss without mentioning, whose extreme generosity will be remembered always.

Abstract

A malicious web content manipulation software can be used to tamper with any type of web content (e.g., text, images, video, audio and objects), and as a result, organisations are vulnerable to data loss. In addition, several security incident reports from emergency response teams such as CERT and AusCERT clearly demonstrate that the available security mechanisms have not made system break-ins impossible. Therefore, ensuring web content integrity against unauthorised tampering has become a major issue. This thesis investigates the survivability of server-side static and dynamic web content using the Web Content Verification and Recovery (WCVR) system. We have developed a novel security system architecture which provides mechanisms to address known security issues such as violation of data integrity that arise in tampering attacks. We propose a real-time web security framework consisting of a number of components that can be used to verify the server-side static and dynamic web content, and to recover the original web content if the requested web content has been compromised. A conceptual model to extract the client interaction elements, and a strategy to utilise the hashing performance have been formulated in this research work. A prototype of the solution has been implemented and experimental studies have been carried out to address the security and the performance objectives. The results indicate that the WCVR system can provide a tamper detection, and recovery to server-side static and dynamic web content. We have also shown that overhead for the verification and recovery processes are relatively low and the WCVR system can efficiently and correctly determine if the web content has been tampered with.

Contents

Publications xv

1 Introduction 1

1.1 An Overview 1

1.2 Research Problem 2

1.2.1 Notions of Integrity 4

1.3 Problem Motivation 4

1.4 Aim of Thesis 9

1.5 Objectives 10

1.6 Contributions of Thesis 10

1.7 Organisation of Thesis 11

2 Background 12

2.1 Introduction 12

2.2 The Web 13

2.2.1 Web Content 14

2.2.2 HTML Legacy Language 16

2.2.3 HTTP Request-Response Model 18

2.2.3.1 Multipurpose Internet Mail Extensions (MIME) . 20
2.2.3.2 Uniform Resource Locater (URL) 21
2.2.3.3 The HTTP Request 21
2.2.3.4 Behind the Scenes of a Web Page 22
2.2.3.5 Response Classes 23
2.3 Web Security 24
2.3.1 Web Security Risks 25
2.3.2 Web Security Issues 26
2.3.3 Malicious Attack 27
2.3.4 Java Security 28
2.3.5 JavaScript Security 31
2.3.6 Common Gateway Interface (CGI) Security 32
2.3.7 Web Security Technologies 34
2.3.7.1 Hashing 34
2.3.7.2 Message Authentication Code (MAC) 36
2.4 Conclusion 36

3 Integrity Verification Systems and Related Work 38

3.1 Introduction 38
3.2 Data Tampering 39
3.3 Survivability 42
3.4 Integrity of Web Documents 43
3.4.1 SSL 43
3.4.2 Digital Signatures 46

3.4.3 Form Field Validation Scheme 47
3.4.4 Network and Application Firewalls 50
3.5 Related Work 52
3.5.1 Web Engineering Security Approach 54
3.5.2 Integrity Verification Systems and Approaches 55
3.5.2.1 Client-side Encryption Approach 55
3.5.2.2 Dynamic Security Surveillance Agent (DSSA) System 56
3.5.2.3 Adaptive Intrusion-Tolerant Server System 57
3.5.2.4 Application-Level Gateway Approach 59
3.5.3 Section Summary and Conclusion 60
3.6 Conclusion 61

4 Design of WCVR System **63**

4.1 Introduction 63
4.2 Web Content Verification and Recovery (WCVR) System 65
4.2.1 Work Assumptions 65
4.2.2 Overview of Web Security Framework Architecture 66
4.2.3 New Model of Interaction Elements 70
4.2.4 New Hashing Strategy 74
4.2.5 Functional Overview 77
4.2.6 Security Framework Components 79
4.2.6.1 Web Register Component 79
4.2.6.2 Integrity Verifier Component 86
4.2.6.3 Response Hashing Calculator 93

4.2.6.4 Integrity Verifier Component at the Response Level 95
4.2.6.5 Recovery Component 96
4.3 Threat Model 97
4.4 Conceptual Comparison 99
4.5 Conclusion 102

5 Implementation of WCVR System and Initial Testing 104
5.1 Tools used for the Implementation 105
5.1.1 Programming Languages for Implementing the Prototype . 105
5.1.1.1 Java 105
5.1.1.2 Servlets 106
5.1.1.3 Java Filters 107
5.2 Architecture Design of the Prototype 108
5.2.1 Web Register Mechanism 110
5.2.2 Response Hashing Mechanism 110
5.2.2.1 Deployment 112
5.2.3 HTTP Interface Mechanism 114
5.2.4 Registry and Integrity Verification using SHA-1 Checksums 117
5.3 Testing Strategy 119
5.3.1 Considerations 120
5.3.2 Formal Experimental Statement 120
5.3.3 Security Testing Strategy 121
5.3.3.1 Network Layout 122
5.3.3.2 Experiment 1 124

5.3.3.3 Experiment 2 126
5.3.4 Performance Testing Strategy 129
5.3.4.1 Experiment 3 130
5.3.4.2 Experiment 4 133
5.3.4.3 Experiment 5 135
5.4 Initial Testing 138
5.5 Conclusion 152

6 System Evaluation **154**
6.1 Introduction 154
6.2 Reflections on Methodology 155
6.3 System Evaluation: Security and Performance 156
6.3.1 Case Study - Security Objective (Detection and Recovery) 159
6.3.1.1 Section Conclusions 164
6.3.2 Case Study for Micro-benchmarking Performance 169
6.3.3 End-to-End Performance Evaluation 172
6.3.3.1 Case Study K 173
6.3.3.2 Case Study R 180
6.3.3.3 Case Study J 184
6.4 Further Discussions and Conclusions 191

7 Conclusions and Future Work **194**
7.1 Research Summary 194
7.2 Conclusions of Thesis 195
7.3 A Summary of Contributions 196

7.3.1 A novel approach to the verification of server-side dynamic web content integrity 196

7.3.2 Improved performance in the verification of static web content 198

7.3.3 The development of the WCVR system to ensure the survivability of web content 198

7.3.4 Experimental studies to evaluate the reliability and effectiveness of the WCVR system 199

7.4 Limitations of Research Work and Future Work 200

A Case Study - Security Objective: Table **201**

B Case Study for Micro-benchmarking Performance: Tables **202**

References **218**

List of Figures

1.1 Snapshot of the requested static code "`viewimage.html`" 6

1.2 Snapshot of the dynamic code "`view.jsp`" at the request level. . . 7

1.3 Snapshot of output response "`view.jsp`" 8

2.1 HTTP request-response model architecture 19

2.2 Request File 23

2.3 Response File 24

2.4 Java Program Architecture 29

3.1 The end-to-end security model and gateway chain model of SSL . 45

4.1 Schematic view of WCVR architecture 67

4.2 Interaction model of a web page 72

4.3 HTML form input element 74

4.4 Example of a web site with 3 web pages and 7 referenced objects . 77

4.5 Algorithm of registration static phase for web contents 82

4.6 Production of MAC value 83

4.7 Extracting of content element for hashing calculation 84

4.8 Producing hash value using SHA-1 hash function and private key (MAC technology) 84

4.9 The finite automata of state protocol 87

4.10 Extraction of *SP* path 89

4.11 Extraction of original hash value 89

4.12 Enforcing request availability policy 90

4.13 This function compares between the original checksum and the re-calculated checksum for integrity verification of static web content 90

4.14 The process of the hashing calculation in our WCVR system . . . 92

4.15 Algorithm of response registration 94

4.16 This function compares between the original checksum and the re-calculated checksum for integrity verification of dynamic web content 95

5.1 Filter Process 107

5.2 High level architectural flowchart of WCVR prototype 109

5.3 The "`web.xml`" for Response Hashing Deployment. 113

5.4 Structure of a HTTPServer Class 118

5.5 The schematic of the network layout for testing purposes. 123

5.6 A list of measurements for the original hash values of web contents 141

5.7 Detecting tampering attacks 141

5.8 Static web content: Graphs for response time (request) curves through DSSA and the WCVR on IIS and Tomcat servers, in seconds, of all requests during the test. 146

5.9 Displays the distribution of page response time, in seconds, of all pages during the test. 147

5.10 Dynamic Web Content: Graphs for response time curves through without verification system and the WCVR on Tomcat, in seconds, of all requests during the test. 150

5.11 Displays the percentage of dynamic pages that were performed within a given time range. Theses graphes help determine the percentage of pages that meet a performance objective. 151

6.1 A sequence of static and dynamic web pages from JSP shopping cart application have been requested. 165

6.2 Log file: `Cart.jsp?itemID=0&count=15` 166

6.3 Log file: `Cart.jsp?itemID=2&count=17` 167

6.4 Log file: `Cart.jsp?itemID=6&count=19` 168

6.5 A linear chart of registry Performance for both SHA1-extended and SHA-1 as compared with file sizes. 171

6.6 Case Study K - Static Web Content: Graphs for the response times (request) through the DSSA and WCVR systems on IIS and Tomcat web servers, in seconds, of all requests during the test . . 175

6.7 Case Study K- Dynamic Web Content: Graphs for response time curves through without verification system and the WCVR on Tomcat, in seconds, of all requests during the test. 179

6.8 Case Study R - Static web content: Graphs for response time (request) curves through the DSSA and the WCVR on IIS and Tomcat servers, in seconds, of all requests during the test. 182

6.9 Case Study R- Dynamic Web Content: Graphs for response time curves through without verification system and the WCVR on Tomcat, in seconds, of all requests during the test. 185

6.10 Case Study J - Static web content: Graphs for response time (request) curves through DSSA and the WCVR on IIS and Tomcat servers, in seconds, of all requests during the test. 187

6.11 Case Study J - Dynamic Web Content: Graphs for response time curves through without verification system and the WCVR on Tomcat, in seconds, of all requests during the test. 190

6.12 Average response time (seconds) on Tomcat and IIS web servers. . 192

Publications

We have published a number of conference and journal papers summarised as follows:

- S. Aljawarneh, C. Laing, and P. Vickers, Verification of Web Content Integrity: A New Approach to Protect Servers Against Tampering, in PGNET 2007 The 8th Annual Postgraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting (M. Merabti, ed.), pp. 159-164, Liverpool John Moores University, 28-29 June 2007, ISBN: 1-9025-6016-7.

- S. Aljawarneh, C. Laing, and P. Vickers, Security Policy Framework and Algorithms for Web Server Content Protection, in Proc. ACSF 2007 2nd Conference on Advances in Computer Security and Forensics (J. Haggerty and M. Merabti, eds.), Liverpool John Moores University, July 2007.

- S. Aljawarneh, C. Laing, P. Vickers, and M. Angelova. Tamper Detection, Prevention, and Recovery in Server-side static and dynamic web content (poster), CEIS Symposium, CEIS , Northumbria University, 2008.

- S. Aljawarneh, C. Laing, P. Vickers, and M. Angelova. Tamper detection, and recovery on server-side web content using a WCVR system: A design, implementation, and experiential evaluation. Will be submitted to Journal of Security and Communication Networks, WILEY.

- S. Aljawarneh, C. Laing, and P. Vickers, Design and experimental evaluation of Web Content Verification and Recovery (WCVR) system: A survivable security system, in Proc. ACSF 2008 3rd Conference on Advances

in Computer Security and Forensics (J. Haggerty and M. Merabti, eds.), Liverpool John Moores University, July 2008.

Chapter 1

Introduction

1.1 An Overview

Ensuring the survivability of server-side web content and the control of the exchanged data has become a major issue. Indeed, a digital nature, web-based textual and multimedia documents can be tampered with and distributed very easily. Therefore, ***the integrity of server-side static and dynamic web content against tampering is still in question.***

In this context, the verification[1] of server-side web content integrity is becoming more important because many web based-services are generated on the fly. It is therefore important to develop systems for integrity verification that are able to provide web content security, detection of malicious manipulation of web content and recovery from tampering.

Cryptography and watermarking seem to be the alternative solutions for reinforcing the security of multimedia documents against tampering attacks (1; 2; 3; 4). When the hidden or encrypted information is exposed (illegally decrypted) by an adversary, the purpose of cryptography and watermarking may be invalid (5). It is difficult to employ end-to-end encryption or end-to-end watermarking

[1] Verification is quality control in which we only take corrective actions. For more on verification, see www.blurtit.com/q252833.html

on all data to achieve end-to-end security because proxies, web servers, and web browsers involve decrypting some or all data to read and even modify some of them to provide client-server services such as executing client and server scripts, transforming and filtering web pages (6).

1.2 Research Problem

Because much academic research focuses on developing a new hashing algorithm or a new encryption algorithm, data integrity has received little attention in information security research and technical security groups (7; 8; 9). Furthermore, there is little published research in methods for testing web content integrity (10). The published research and technical communities in the web security area are generally more concerned with cryptographic rules and algorithms. In an attempt to remedy this, this thesis focuses on the integrity of data. If the integrity of data is violated, its confidentiality and availability can be compromised. It should be noted that data integrity refers to the trustworthiness of information resources, thereby ensuring that only an authorised client can alter the data – unauthorised tampering may result in incorrect or malicious web application behaviour behind installed firewalls (11; 12; 13).

Adversaries can evade a server-side web content by using malicious web content manipulation software. Static and dynamic web server content can be tampered with by changing (i) the style classes such as Cascading Style Sheet (CSS), (ii) referenced objects (images, audio, video, and other objects) or (iii) the source code of the web page itself through running malicious code on the server that compromises a requested page before the client receives it (4; 7; 8; 9; 11; 14). For example, it is possible to replace an original image by another image containing malicious code. A victim requests the altered image and then it can disrupt the contents of a web server or client machine. In addition, the CSS object is threatened through a visualization spoofing attack. The strategy of this attack is to change any important information that is identified by a particular colour to another colour. The objective of this attack is to manipulate the user into making a decision that is based on incorrect information (9; 11; 14).

Users might notice the alteration of web content after the authentication scheme[2] has been performed. However, at this stage the destruction of web content has already taken place (7; 12). The integrity of web content can be violated on the server even though the communication channel between the server and client is secure.

Ensuring static and dynamic web content integrity against unauthorised tampering, even when the communication channel between client and server-sides is protected, has become a major issue. ***The question this thesis addresses is how the integrity of server-side static and dynamic web content can be verified against the tampering attacks before the client receives the requested page. As a part of problem, we have tried to solve the issue of data recovery if web content has been tampered with.***

The integrity of a web server environment depends on dynamic, unstructured data that is generated by running server-side scripting dynamic web pages (15). The key issue in this thesis is that even if the application knows that this data can impact on applications integrity, the hashing measurement of static web content on the repositories of a web server is useless because we cannot predict values that would preserve integrity. In addition, even though the server-side scripting dynamic web pages are verified, the integrity of generated dynamic web content can be tampered with.

In a web server, the key dynamic data are (15):

1. The various types of requests from remote clients, administrators, and other Servlets and
2. Database Management System (DBMS) tables.

To provide integrity verification services, we first examine the meaning of data integrity, in general.

[2]The process of verifying the account credentials of systems or users.

1.2.1 Notions of Integrity

In the security context, an integrity objective is clearly defined as one which ensures that the sent and received data are identical. It means, ensuring that data have not been modified, whereas modifications include deletion or alteration of existing data and each addition of new data to existing data (16). This binary definition (i.e. the data either has integrity or it does not (16)) can also be applicable to any type of web content such as textual and multimedia content. Indeed, in real life situations, web content can be transformed into different formats and results. In other words, modifications to a web document may change its meaning or visually degrade it. In order to provide verification of integrity for web content, it is important to distinguish between malicious manipulations, which consist of changing the content of the original web content, and manipulations related to the use of an image or audio, such as format conversion, compression, filtering, and so on. Unfortunately this distinction is not always clear, it partly depends on the type of image or audio and its use (1; 2; 3; 11; 17; 18).

1.3 Problem Motivation

Our motivation for this thesis is sixfold:

1. Organisations that rely on information systems as the primary way to conduct sensitive transactions are increasingly concerned about their reputation when web systems are subverted (7; 19; 20; 21). The Computer Emergency Response Team (CERT[3]) (22) has reported a dramatic increase in the number of security vulnerabilities[4] which threaten web content (5990 in 2005 to over 7000 in 2007).

[3]A centre of Internet security expertise, located at the Software Engineering Institute, a federally funded research and development centre operated by Carnegie Mellon University.

[4]Weaknesses in a computing system that can result in harm to the system or its operations, especially when this weakness is exploited by a hostile person or organisation or when it is presented in conjunction with particular events or circumstances.

2. The Secure Sockets Layer (SSL) protocol (23) was developed to support the integrity of data transit (7; 8; 9; 12; 24) and can provide a secure point-to-point channel. However, it has problems in the presence of application gateways (6). This means that cryptographic security protocols, such as the SSL protocol, do not provide a complete solution to tackle the tampering attacks and must be complemented by additional protection mechanisms.

3. The current security technologies such as firewalls, Intrusion Detection Systems (IDSs), Intrusion Prevention Systems (IPSs), cryptography, and access control are not capable of verifying the integrity of web content before a request or response enters the secure communication channel (7; 12; 13; 25). A technical mechanism alone does not provide a standard policy and cannot distinguish between the original HTTP (Hyper Text Transfer Protocol) conversation and the altered HTTP conversation (8). In addition, many are designed at the network/host layer, not at the application layer level (25). Furthermore, tampering attacks can take place behind firewalls (4; 8). Therefore, there is no one-stop-shop security method that meets all the security requirements and design specifications of new or existing web applications.

4. HTTP is sessionless (24; 26; 27; 28). The integrity of web content relies on the integrity of the HTTP Request-Response model. Therefore, if this model fails, the data integrity may be violated (9; 29). This model can fail because web servers and web browsers do not properly manage the statelessness of HTTP, in which each client requests results in a new connection between a web browser and a web server. The Common Gateway Interface (CGI) supports the maintenance of state through the use of hidden variables or cookies that keep track of the current information for each request (28; 30). However, it is possible to save the HTML form, modify the hidden values of its fields, and then reload this altered form into a web browser for rendering (28; 31; 32). Zhou (26) has identified some problems with web server models including web server models cannot ensure the security of continuity of HTTP conversations on a server – they are more concerned with

the implementation of the cryptographic rules than the implementation of a security analysis of the system's functions.

5. Dynamic data is a critical issue. The generation of dynamic web content depends on user interaction. Different user information leads to different generated web content. Therefore, it is very difficult (even impossible) to analyse the requested page of dynamic code before processing on a web server (see Figure 1.1 and Figure 1.2). The dynamic code of server programming languages needs to be processed on a web server before returning the response to a web browser. As a result, we cannot guarantee that dynamic code is not tampered with even if the static code is verified; therefore the generated web content should also be verified.

```
1: <HTML><HEAD><TITLE> Preview </TITLE></HEAD><BODY>
2: <FORM><CENTER><B> Image 1 </B>
3: <IMG SRC=http://www.xyz.com/images/image1.gif/>
4: <INPUT TYPE=button VALUE=Close/></CENTER>
5: </FORM></BODY>
6: </HTML>
```

Figure 1.1: Snapshot of the requested static code "`viewimage.html`".

In Figure 1.1, the web browser requests a static document such as an HTML file "image.html". The web server locates the resources in their designated directories and serves the requested file back to the web browser. Requesting a document and responding to a request are defined by HTTP (23), which forms the basis of client-server interactions on the Internet. The Web server completely processes the request and there is no need for interaction with a Servlet (33) container or any web application server because the static HTML file does not generate dynamic web content. Therefore, it is easy to understand the code of a static file at the request level.

In Figure 1.2, the web browser requests a web page containing a Java Server Pages (JSP) page "view.asp". The JSP pages generate dynamic contents on the fly. The Web server forwards the request to the JSP web application server and backend database to process and gets the value of `fileName` (see line 9 in Figure 1.2). After that the web server returns a static HTML file (see Figure 1.3) to the web browser for rendering. In Figure 1.2, it is impossible to know the name of the resource (image file name) at the request level before processing on the web application server and backend database.

```
1:<\% String name=request.getParameter("customerName");
2: DBConnection myDB= DBConnection.getDBConnection();
    try  {
3:    myDB.createConnection();
4:    Connection conn=myDB.getConnection();
5:    String sql= "select filename from file_information where name='" +
6:    fullName + "'  ;
7:    PreparedStatement  stmt = conn.prepareStatement(sql);
8:    stmt.executeUpdate();
9:    fileName=stmt.getString("filename")
10:   stmt.close(); }
      catch(Exception ex) { ex.printStackTrace(); }
      finally {myDB.finalize(); }
11: write("<HTML><HEAD>");
12: write("</HEAD><BODY");
13: write("<FORM ACTION=../Thank.jsp>");
14: write("<IMG SRC='" + fileName + "'>/");
15: write("<INPUT TYPE=button/>");
16: write("</FORM>");
17: write("</BODY>");
18: write("</HTML>");
19: close();\%>
```

Figure 1.2: Snapshot of the dynamic code "`view.jsp`" at the request level.

```
1: <HTML>
2:<HEAD></HEAD>
3:<BODY>
4: <FORM ACTION=../Thank.jsp>
5: <IMG SRC=johnsmith.gif/>
6: <INPUT TYPE=button/>
7:</FORM>
8:</BODY>
9:</HTML>
```

Figure 1.3: Snapshot of output response "`view.jsp`"

6. Web applications[5] often have direct access to backend databases and, hence, sensitive data is much more difficult to secure (37). If there is no direct access to backend databases, attacks can use legitimate application protocols such as HTTP, and Simple Object Access Protocol (SOAP) to capture data and transmissions (25; 37; 38). Access to databases through Web browsers is now common place with some form of web server being provided by all major vendors of database software (39). e-Commerce, online banking, enterprise collaboration, and supply chain management sites have concluded that at least 92% of web applications are vulnerable to some form of attack (40). The Gartner[6] study found that 75% of Internet assaults are targeted at the web application level (25).

   Web applications such as shopping carts, login pages, dynamic content, and other applications have been designed to allow web site visitors to

[5]A web application is a collection of integrated static and dynamic web pages on a web system. The web application is run on a web browser, a web server, or both (34; 35). It is organised into three tiers: a web browser tier, a web server tier, and a backend database tier. The user interaction is proposed in a web browser tier, the program logic (such as ASP and JSP) is run in a web server tier, and the data operations (such as addition, deletion, and updating) are performed in a database server tier (35; 36).

[6]Gartner Group is an information and technology research and advisory firm headquarteres in Stamford, Connecticut. It was known as The Gartner Group until 2001. The group consists of Gartner Research, Gartner Executive Programs, Gartner Consulting and Gartner Events. For more details, visit http://www.gartner.com/.

retrieve and submit dynamic content including varying levels of personal and sensitive data (37).

1.4 Aim of Thesis

The aim of this thesis is to investigate the survivability of server-side static and dynamic web content using a Web Content Verification and Recovery (WCVR) system. In this thesis, the survivability is the capability of a web content to continue its mission over the HTTP request-response model even in the presence of illegitimate modifications to a web content. The question then arises, what happens when an altered web content has been detected? Our survivability strategy in the proposed WCVR system can be set up in two steps:

1. Detection and response by integrity verification process.
2. Recovering from tampering attacks by recovery process.

Our approach is applicable to all kinds of tampering attacks and processing of all data types on the server-side. For example:

- visualisation spoofing attack.
- textual spoofing attack.
- web application verify.
- tampering code manipulation (source code, path, and link).
- tampering object manipulation(audio, images, video, and other referenced objects).
- defacement of web page.

1.5 Objectives

1. Review the existing integrity verification systems and related work, and consider their strengths, weakness, and limitations.
2. Develop work assumptions, web security policy framework, and models.
3. Develop a web security system architecture and supporting software system to verify static and dynamic web content on the server before the client receives the requested page.
4. Conduct a series of experimental studies and evaluation of the system.
5. Report experimental outcomes.

1.6 Contributions of Thesis

We have summarised the major contributions of thesis as follows:

- A novel approach to the verification of server-side dynamic web content integrity.
- A novel approach to the recovery of server-side dynamic web content.
- Improved performance in the verification of server-side static web content.
- Improved performance in the recovery of server-side static web content.
- The development of the proposed WCVR system to assist in the survivability of server-side static and dynamic web content.
- Experimental studies to evaluate the reliability and effectiveness of the WCVR system.

1.7 Organisation of Thesis

This thesis is organised into seven chapters as follows:

- Chapter 2 describes the elements of web system, web content, and the data flow over HTTP Request-Response model. It also identifies the web security definitions, issues and technologies.
- Chapter 3 identifies the tampering attacks of static and dynamic web content, and presents the limitations, requirements, strengths and weaknesses of existing systems and approaches.
- Chapter 4 describes the design of proposed integrity verification and recovery system (threat model, work assumptions, models, and framework), and compares the work in this dissertation to prior and related work.
- Chapter 5 presents the implementation of WCVR system and the proposed mechanisms. The tools used in creating the prototype are discussed and the architecture of the prototype is depicted. In addition, the components of the prototype and their functions are explained and how the components communicate each other is clarified. Chapter 5 also describes the experimental design, and pilot study. We have designed five experiments to meet the security and performance objectives.
- Chapter 6 shows and discusses results of experimental studies, and performance evaluation. We have carried out five case studies for evaluations purposes.
- Chapter 7 draws conclusions and discusses possible future work.

Chapter 2

Background

2.1 Introduction

Computer and web security are critical issues over HTTP request-response model (21; 41; 42). The infrastructure of networks, routers, domain name servers, and switches that glue these web systems together could fail, and as a result, web systems will no longer be able to communicate accurately or reliably. A number of critical questions arise, such as what exactly the infrastructure is, what threats it must be secured against, and how protection can be provided on a cost-effective basis. Underlying all these questions is how to define a secure web system (42).

Cryptography and watermarking seem to be the alternative solutions for reinforcing the security of multimedia documents against tampering attacks (1; 2; 3; 4). When the hidden or encrypted information is exposed by an adversary, the purpose of cryptography and watermarking may be invalid (5). It is difficult to employ end-to-end encryption or end-to-end watermarking on all data to achieve end-to-end security because proxies, web servers, and web browsers involve decrypting some or all data to read and even modify some of them to provide client-server services such as executing client and server scripts, transforming and filtering web pages. However, if sensitive data is decrypted to untrustworthy proxies,web servers, and web browsers for processing, it can result in information leakage and tampering (6).

This chapter will present the description of web content and the HTTP request-response model in Section 2.2. In Section 2.3, it will discuss the web security issues, objectives, and technologies. Placing web security in perspective is important because it is a central issue and necessary to organisations, clients and even home users now and in the future. Indeed, what is web security? But few people realise it exactly. Core to web security are the issues of confidentiality, integrity and availability which refer to keeping data secret, ensuring data remains intact and ensuring systems are responsive.

2.2 The Web

In the late 1960s, the Advanced Research Projects Agency (ARPA) sponsored a project for implementing the ARPANET, the legacy of the Internet. The main purpose of the ARPANET project was to allow multiple users to make request and response messages simultaneously over the same communication channel via phone lines. The information was divided into a number of packets and then routed to their destinations. Each packet consisted of sender address, destination address, additional information for checking the integrity of communication, and part of the data. The communication protocol, which was used in ARPANET project, is called Transmission Control Protocol (TCP). The aim of this protocol is to ensure that the messages are correctly routed from sender to receiver over the communication channel of the ARPANET system (30; 34).

However, some challenges arose such as how to communicate across a network of networks. ARPANET improved the TCP protocol to be the Internet Protocol/Transmission Control Protocol (IP/TCP) protocols. Currently, they are the basic architecture of the Internet (30; 34).

Subsequently, Berners Lee (43) developed World Wide Web (WWW) at the CERN as a medium for the broadcast of read-only material in 1990, as well as the concept of hypertext. The WWW is a distributed environment that allows users to communicate and view multimedia-based documents over the Internet (13). In 1993, web-based services were explored by the Mosaic browser, which

had a graphic Interface. Currently, most major web browsers (such as Microsoft Internet Explorer, Netscape Navigator, and Firefox) are used to explore the web-based services (27; 30).

The WWW Consortium (W3C) was founded in 1994 to make the web universally accessible and available regardless of ability, language, or culture (30). W3C provides the automatic online validation service, which is free of charge and enables web managers to test and correct their Hyper Text Markup Language (HTML), Extensible Hyper Text Markup Language (XHTML), Cascading Style Sheet (CSS) and Extensible Markup Language (XML) documents (30; 44).

Web technology has introduced a new distributed computing paradigm. This is suitable for various web oriented applications, web administrative applications and general web applications including e-Commerce, e-Banking, e-Shopping, e-Ticketing, e-Finance, and e-Management (7). Web technology has incorporated database connectivity to be able to access much information in an online state. This information is stored on a server using DataBase Management Systems (DBMS) database application for processing (30).

2.2.1 Web Content

Web content is a textual or multimedia content that is encountered as part of the user experience on web sites. It includes text, images, sounds, videos, objects and animations (7; 45; 46).

The use of hypertext concept, hyperlinks concept and a page-based model of sharing information help to define web content, and to form the architecture of web sites. Currently, the categorisation of web sites is based on a type of web site where web content is dominated by the page concept (44). When an address is requested, such as `http://www.google.com`, a range of web pages are viewed, but each page could have embedded tools to view video clips or other data types (30; 43; 46).

For example, e-Commerce sites could contain textual material and embedded graphics displaying a picture of the item(s) for sale. However, there are few sites

that are composed page-by-page using some variant of HTML. Generally, web pages are composed as they are being served from a database to a customer using a web browser. However, a user sees the mainly text document arriving as a web page to be rendered in a web browser (46).

Web content consists of dynamic and static data. Dynamic web content is the content (text, images, form fields, etc.) that can change on a web page in response to different conditions such as user interaction (30; 47). There are two ways to create this kind of interactivity (48; 49):

1. Using client-side scripting to change interface behaviours within a web page, in response to mouse or keyboard actions or at specified events. In this case the dynamic behaviour occurs within the presentation of a web page.

2. Using server-side scripting to change the supplied page source between pages, adjusting the sequence or reload of web pages or web content supplied to a web browser. Web server responses may be determined by such conditions as data in a posted HTML form, parameters in the (Uniform Resource Locator) URL, the type of web browser being used, the passage of time, or a database or server state.

The result of either technique is described as a dynamic web page, and both may be used in parallel. In the first appraoch of interactivity, web pages must use presentation technology called rich interfaced web pages. Client-side scripting languages such as JavaScript used for Dynamic HTML (DHTML) and flash technologies, are normally used to activate media types (sound, animations, changing text, etc.) of the presentation. The scripting also allows use of remote scripting, a technique by which the DHTML page requests additional information from a server, using a hidden frame, `XMLHttpRequests` object, or a web service (30; 47).

Web pages that adhere to the second approach are almost generated with the help of server-side languages such as Active Server Page (ASP or ASP.NET), Java Server Page (JSP), etc. These server-side languages typically use the Common Gateway Interface (CGI) to generate dynamic web pages (30; 47).

The client-side dynamic content is generated on a client's machine. A web server retrieves the page and sends it as is. A web browser then processes the code embedded in the page and displays the page to a user. However, some users have scripting languages disabled in their web browsers due to possible security threats. In addition, some web browsers do not support the client-scripting language or they do not support all commands (such as `write` command and `innerHTML` property) of the language (47; 50).

Server-side dynamic content is a little bit more complicated. The following steps illustrate how the server-side dynamic web content is produced.

1. A web browser sends an HTTP request.
2. A web server retrieves the requested script or program.
3. A web server application executes the script or program which typically outputs an HTML web page. The program usually obtains input from the query string or standard input which may have been obtained from a submitted web form.
4. A server sends the HTML output to a web browser.

Another type of content is referred to as static. A static web page is a web page that always comprises the same information in response to all downloaded requests from all users (48). The most obvious requests are the transmission of images and blocks of text. This is the data that is found on virtually every first page in a site and which forms the basis of virtually every page. This type of page is usually called "static HTML". In the next section, we describe the structure of HTML web page.

2.2.2 HTML Legacy Language

HTML is a legacy language of web technology. Each electronic document (or web document) contains a predefined set of HTML tags that might embed active

content modules. The page is accessed by URL. HTML technology supports multimedia documents including video, sound, text and dynamic links as well as the textual interface (13; 27; 30).

A web document is platform-independent, based on HTML language and its successor is XML (13; 30). It can be viewed by various web browsers on various operating systems. HTML is a content mark-up language that a web browser uses to interpret and display web documents.

HTML analysers and Script analysers process web documents. The HTML analyser is to process the HTML tags, while the Script analyser is to process the embedded scripts either in the client-side or in the server-side (35).

HTML documents consist of two sections: head and body (51). Each section includes HTML elements and HTML sub-elements to describe a web layout. HTML supports form element `<form>` that permits user interaction. This element contains sub elements such as `<input>`, `<select>`, `<textarea>` and others (52). A HTML form has two fundamental functions (13; 53; 54):

1. Providing area on a web page to enter a particular data that is sent to a web server for processing.

2. Allowing validation of input data by invoking `script` element which resides on a web document.

HTML also includes `script` element that was provided by Netscape 2.1 and beyond for data input validation. This element supports powerful scripting languages (such as JavaScript and Visual Basic script) to perform interactive tasks. One of the main tasks of scripting language is to check the user input errors on the client-side rather than on the server-side, because if a server finds any input error, a server returns an error message to a web browser, therefore, the client-side validation modules saves round-trips over a network (36; 54).

Furthermore, `<applet>` and `<object>` HTML elements provide a link to embed Java Applets, multimedia objects, and ActiveX objects (30; 51). For example,

a web browser downloads an ActiveX object after rendering using `<object>` element to run inside a web page. However, ActiveX presents a vulnerability to the client because the ActiveX object is not restricted and can directly access operating system resources on a client machine. However, Microsoft Internet Explorer verifies the ActiveX object using Authenticode technology (27). This technology is supported by the digital signatures and the Public Key Infrastructure (PKI). It can sign the ActiveX object, which is run in a machine code or in a Java Bytecode.

Because it does not support specific HTML validation tags, HTML is not a specific high level domain language to validate the user interaction. In addition, HTML does not offer the extensibility of own user tags and attributes (52; 55). HTML is not also suitable for complex data entry that consists of many forms (56). Therefore, W3C has developed the XML language. It is a purely declarative and high-level domain language (52). This means that XML supports extended tags and attributes for validation modules without the need for any programming skills.

2.2.3 HTTP Request-Response Model

HTTP request-response model is constructed from three parts: a web browser, a web server and a communication channel between a client and a server (26; 27; 44; 57). Figure 2.1 illustrates the components of HTTP request-response model architecture.

The web browser is a software application that is used to access WWW and has three basic functions (30; 44):

- Obtaining information on the Internet using the URL and communicating using HTTP.
- Rendering HTML source code that is receiving in the form of HTTP response from a web server.

- Current web browsers provide Graphic User Interface (GUI) tool for performing different tasks such as saving web documents, searching, and others.

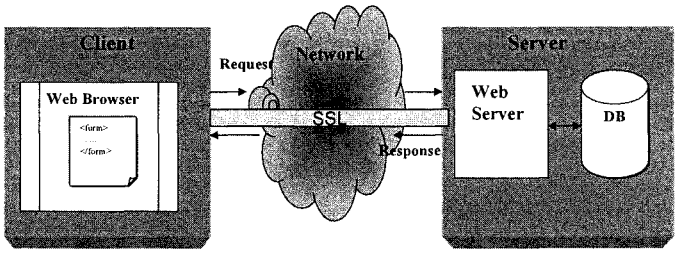


Figure 2.1: HTTP request-response model architecture

A web server is an independent platform (13) that is structured from software and hardware. However, a web server has several basic functions (13; 44; 58):

- Logging activities.
- Authenticating users.
- Responding web documents to authorised users.

All HTTP operations are based on the HTTP request-response model. In a web environment, a request is an operation to be performed on URL. Meanwhile, a response is defined as web server answers or replies for request operation to a web browser (44).

The data integrity relies on the integrity of the HTTP request-response model. Therefore, if this model fails, then the data integrity may be violated (9; 59). A user can access a client through a web browser to make a request by clicking on hyperlink text or hyperlink image, clicking on a submit button or command button, redirecting using the `action` attribute of `form` element, or by setting the requested URL at the address bar of a web browser (9; 44; 59).

Data is sent to a web server via the communication channel. Communication channels are normally secure and controlled by some communication protocols

including TCP/IP, and SSL (27). The aim of a web server is to manage and control all user requests, communicate with the correct user, save the request information on a server database, and then control the access of user to web page resources. For sensitive web content, a web server authenticates a request that contains a username and a password and then:

- Accept the request, and then allow the connection if they are correct, otherwise,
- Refuse the request and then close the connection.

This request is processed on a web server by an application written in a server-side programming language. As a result, a data query may save data into a backend database. Finally, a web server sends the appropriate web page to a web browser, which renders it (27; 44).

2.2.3.1 Multipurpose Internet Mail Extensions (MIME)

Multipurpose Internet Mail Extensions (MIME) is a scheme that lets electronic mail messages contain mixed media (sound, video, image, and text). The WWW uses MIME content-types (text/html, application/html, text/html-external-parsed-entity, image/gif, etc.) to specify the type of data contained in a file or being sent from an HTTP server to a client (60).

Each web browser has a different configuration for mapping the types of data to particular function. Major web browsers can process various types of HTML documents, and CSS but other types are sent to various programs via the plug-in mechanisms such as sound player, video player (27; 30; 51; 60).

The six MIME types defined by the RFC are as follows (60):

1. Text.
2. Message.
3. Application.

4. Image.
5. Audio.
6. Video.

2.2.3.2 Uniform Resource Locater (URL)

Uniform Resource locator (URL) is a method of addressing web documents in the WWW. URL consists of (13; 44; 61):

1. Protocol (such as HTTP).
2. Host name.
3. Internet port number of service. If not specified, the default port number (80 for HTTP) is used.
4. Location of resource on a server (path/query).

The URL components can be specified in HTML/XHTML `form` element through `action` attribute. The URL can be specified in a `Link` element through `HREF` attribute of `A` tag.

2.2.3.3 The HTTP Request

There are three parts of HTTP request operation: a request line, request header and request body (this part is optional depending on the form author) (27; 44; 51). The request line starts with request method, followed by a resource identifier and the protocol version, For Example:

- Get/default.html HTTP/1.0
- Post/index.html HTTP/1.0

The `form` element has a number of methods to send the user input from a web browser to a web server as follows:

- GET: is the default request type, which notifies a web server to fetch a document and send it back to a web browser. It is normally used to retrieve data such as search engine or data query. This type appends the form information directly to the end of the URL. The input field name and input field value are represented as a pair of parameter name and its value at end of URL after the question mark (44; 61; 62), as shown in Table 2.1.
- POST: is used to process any kind of data in various HTML form services such as sorting, updating, ordering a product, sending e-mail, or responding to a query (44).
- HEAD: is used to retrieve header information of a web document such as a version of a document, and availability of hyperlinks (44; 61).
- DELETE: is used to delete a recourse identified by a URL during the need of a web server (44; 61).

For utilization of request operation, a user can send additional information about a web browser and user itself. This additional information are called request header fields such as Accept, Encoding, Authorization, Authentication, Host and User Agent (7; 44; 61).

2.2.3.4 Behind the Scenes of a Web Page

The example in Table 2.1 shows a simple web page that contains a list of hyperlinks of universities in United Kingdom. A web server responses and sends back the HTML file and the header file to a web browser. A web browser parses this URL $http://www.findaschool.org/index.php?Country = United + Kingdom.$ Table 2.1 shows meaning of each part of URL (27; 61):

Table 2.1: Parts of URL

Content	Meaning
http://	Hyper Text Transfer Protocol.
WWW	World Wide Web
findaschool.org	Host name.
index.php	A document path in a server.
?Country=United+Kingdom	A query path, which contains the parameter names and their values. The values of parameters are URL-encoded: space becomes +, non-alphanumeric chars become %hexcode for encryption.

A web browser connects to `findaschool.org` using the HTTP protocol. The default port for HTTP is 80 if it is not specified. Figure 2.2 shows the structure of request message (62):

```
GET/ HTTP/1.0 (Request Get, HTTP protocol, Protocol version 1.0)
Connection: keep-Alive (TCP Connection is open until to disconnect)
User-Agent: Microsoft Internet Explorer 6.0 (Win XP)
Host: findaschool.org- (hostname on server)
Accept: image/gif, */* (>Media Type to be accepted)
If-Modified-Since: Friday, 10-Feb-06 11:12:30 GMT (last modified)
```

Figure 2.2: Request File

A web server returns the header file "header information" and HTML file to a web browser as shown in Figure 2.3.

2.2.3.5 Response Classes

The response classes rely on the status code in a header file. In respect to the first digit of status code, the type of response classes is determined as follows:

(44):

1. 1xx: (Informational) request received, still in process.
2. 2xx: (Success) the operation is successfully received, parsed and accepted.
3. 3xx: (Redirection) further process required for competing the request.
4. 4xx: (Client Error) the header file of request contains syntax error and cannot be performed well.
5. 4xx: (Server error) the server failed to perform a valid request.

```
HTTP/1.0 200 ok (Protocol version, status code)
Date: Sat, 10 Feb 2006 13:00:10 GMT (Current Time on the Server)
Server: Apache/1.1.1 (type of software running on server)
Content-Type: text/html (type of document that being sent to web browser)
Content-Length: 327(size of document-byte unit)
Last-modified: Sat, 10 Feb 2006 13:30:10 GMT ->the recent modification time
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 STRICT//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html><head>
<title>GEO | Colleges and Universities | United Kingdom</title>
...
</head><body>
...
</body></html>
```

Figure 2.3: Response File

2.3 Web Security

Any discussion of web security necessarily starts from a statement of requirements (i.e. what it really means to call a web system secure.). Generally, secure systems will control, through use of specific security policies, access to information such

that only properly authorised individuals, or processes operating on their behalf, will have access to read, write, create, or delete information (63).

Organisations that have a web presence are increasingly worried for for their reputations if the web system is subverted. This is because current security tools may not prevent the web system vulnerabilities (8; 17). For example, with 7,247 vulnerabilities disclosed in 2006, total vulnerability count increased nearly 40% over the previous year. This trend of increase is expected to continue (64).

Stein (31) outlines a number of definitions of web security from the user's perspective. For some, web security is the ability to view Internet content in peace and safety. For others, it is the ability to conduct safe business and financial transactions. For web authors, it is the confidence that individuals will not damage their web sites.

2.3.1 Web Security Risks

Users and organisations are suffering from a growing number of attacks that abuse data on the Internet. These attacks become a source of threat for a web system (8; 17; 65). Stein (31) explains that an organisation that has established a strong reputation may suffer from persistent destruction to its reputation after an attack.

Web security vulnerabilities result from poorly configured operating systems, limitations of web servers and web browsers, weakness of web technologies and weakness of software engineers (66). There are three types of security risk (26; 27; 31):

1. Source code problems in a web server that enable an adversary to :
    - Read and copy confidential documents on a web server and this causes loss of data confidentiality.
    - Run malicious code on a server. This permits tampering with web documents and subsequent loss of data integrity.

- Obtain header response information from a web server that will permit breaking into web system recourses. As a result, there is a loss of data availability.
- Activate denial-of-service attacks that might break into the web system.

2. Browser-side risks including:
   - Enable active content vulnerabilities on a web browser such as malicious Java Applet attack, and script attacks.
   - Loss of data integrity at the form level.
3. Network-side risks that cause the eavesdropping problem (i.e. listening to someone else's conversation (67)) from any point including:
   - A web browser connection.
   - A web server connection.
   - Communication channel of a web system.

2.3.2 Web Security Issues

Studies and surveys indicate that web security objectives are easily violated in a web environment (8; 17; 68). There are three basic security issues (8; 17; 69; 70; 71):

1. Data integrity: The data integrity objective ensures that the data can only legally be altered by an authorised client. Data integrity is considered in this thesis.
2. Data confidentiality: The data confidentiality objective refers to limiting information access and disclosure to authorised users. Confidentiality involves limiting the disclosure of data by ensuring that it only becomes known to authorised users. A common method to ensure confidentiality is to encrypt data (known as plaintext) with a secret key known only to the authorised

users to produce encrypted text known as the ciphertext. A good encryption scheme will make it difficult or impossible to recover the original data from the encrypted data without knowledge of the key. Data confidentiality is not considered in this thesis.

3. Data availability: The data availability objective refers to information system that is not available when you need it. Data availability is not considered in this thesis.

In information security, data integrity refers to the validity of data. Data integrity can be compromised through (7; 8; 17; 70):

1. Malicious modification, such as an adversary altering an account number in a bank transaction, or forgery of an identity document.
2. Accidental modification, such as crashing hard disk of a web server.

The privacy and security issues (confidentiality, integrity, and availability) are exploited for the lack of a web browser and limitation of web technologies (66). For example, Java is the safest security model because it supports sandbox class model to protect the Java Applet from some potential security vulnerabilities. Furthermore, Java Applets do not support direct or indirect connection to the operating system environment. On the other hand, JavaScript and ActiveX is the least secure model. Indeed, JavaScript supports methods to capture user details (confidentiality issue) by reading the user files. Integrity problems might result from JavaScript methods that alter or destroy user details. While data availability problems result from JavaScript methods that destroying the user session by reading the cookie details or running an infinite loop to open infinite number of windows and then crash the operating system (66).

2.3.3 Malicious Attack

A malicious attack in the web context is any code launched to disrupt or harm an intended request-response conversation on a web system, including attack scripts,

Java Applet attack, input validation attacks and ActiveX control attack (72). A malicious attack causes disruption to the HTTP request-response operations, or causes eavesdropping problems (72; 73).

Due to the insecurities of the TCP/IP communication protocol, the open architecture of the Internet and the lack of web security mechanisms, the web environment is vulnerable to security risks (34).

- Analyse the server architecture, server type, operation system types and limitations of web browsers.
- Study the web environment architecture such as studying the HTTP Request type (GET, POST, and DELETE) and user input interaction. A criminal searches for security holes in a feedback form, an inquiry form or a login form.
- Study input validation modules. These modules determine whether a certain form data is safe or unsafe data is rejected during the validation processing.

In data integrity, malicious data corruption is more difficult to counter. A simple hash function such as MD5 is insufficient because hash functions are assumed to be public knowledge and easy to compute. An adversary would still be able to corrupt the data provided they also recomputed the corresponding hash. An alternative is to encrypt the hash value with a secret key to form a Message Authentication Code (MAC). Only authorised parties own the key and can generate and verify matching MACs. It is possible to use a hash function to build a MAC using an algorithm such as HMAC (74; 75). In this thesis, we have attempted to address only the data integrity issue.

2.3.4 Java Security

Much research (7; 61; 76; 77; 78) has proved that the Java and JavaScript languages are the most popular implementation of Applets and scripts respectively.

Java was designed to work in a network-computing environment such as downloading Applets over networks. Currently, Java is a safe programming language that protects users from some security vulnerabilities. In addition, Java supports the code signing using JDK 1.6 for securing the integrity of Java Applets (27).

The existing Java security policy contains a number of classes to limit what a downloaded web page can do, including the `Sandbox` class, `SecurityManager` class, `Bytecode Verifier`, and `Loader` class (27). When Java programs are launched, a `Bytecode Verifier` is run to validate for any unauthorised operations. `Bytecode` is a name given to machine code for compiling a Java program in the Java Virtual Machine (JVM) and then interpreted on the running machine (73; 79). Figure 2.4 illustrates how to compile Java program and then download on a web system.

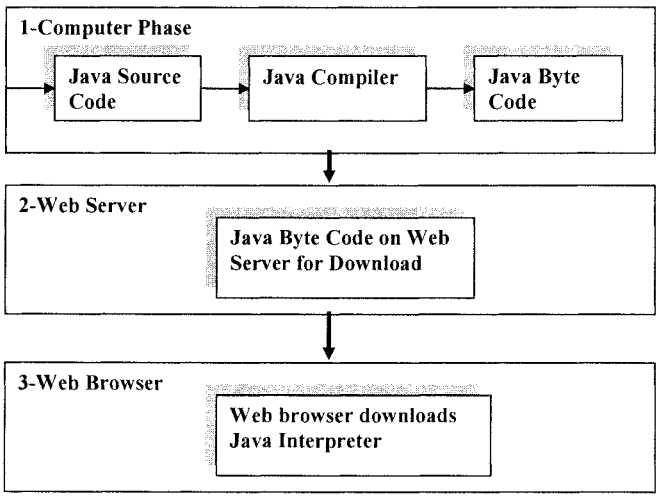


Figure 2.4: Java Program Architecture

Java includes JVM platform to provide a trusted environment for running the Applets, which are embedded in a web page. However, JVM cannot secure a web page against a malicious Applet (80). McGraw and Felten (81) define a malicious applet as "Any applet that performs an action against the will of the user who invoked it should be considered malicious". It is important to

understand the architecture of java security model to know how to detect the Java Applet vulnerabilities (82). The Java policy in JVM does not prevent untrusted Applets from disrupting the users (79).

The default security policy of the Applet sandbox prevents from (79):

- reading and writing files on the client file system.
- making network connections except to the original host.
- creating listening sockets.
- starting other programs on the client system.
- loading a new dynamic library.

There are two attack techniques that cause to loss of data integrity as follows (79):

1. Through type confusion: Because Java is a type safe language, any conversion between data items of a different type must be performed in an implicity way. This type occurs in a result of a flaw in one of the JVM components, which creates the possibility to perform the cast operations from one type to any unrelated type in a way that violates the Java type casting rules. In a result of this attack, Java language security can be circumvented. For example, private, public and protected access is no more important.
2. Through bad implementation of system classes: Any flaw in the implementation of a system might expose some restricted functionality of the native operation system to the untrusted code.

Java supports a Servlet application to work over a web system. Servlet is used to generate dynamic web pages on demand (each time the page is requested) (27). The contents of a dynamic web page can be different each time at each request, because the construction of dynamic page relies on a user interaction.

The Servlet includes `HTTPservlet` class that manages the multiple requests. This feature solves the sessionless HTTP drawback (28).

A Servlet was developed instead of CGI for better scalability and security (36; 38). A Servlet structure contains HTML tags that are embedded in the Java source code. It has full access to HTTP Layer and has the ability to trace each request-response conversation (61). However, the change of web page needs to change in Java code rather than HTML code. As a result, a designer needs to be familiar with Servlet structure or Java Programming language (38).

2.3.5 JavaScript Security

JavaScript is an object-oriented language that was developed to make animation, form field validation and other interactive features (27; 83). JavaScript programs reside in HTML files, usually surrounded by `<script> </script>` in a web document. A web browser can render JavaScript commands in a HTML file. JavaScript supplies additional objects (83):

- Client-side JavaScript: JavaScript supplies objects to control a web browser and a Document Object Model (DOM). For example, an object can be extended to have some user-defined methods. This object is invoked by an event-handler on a form element including on click and on mouse over.
- Server-side JavaScript: extends by supplying objects to run on a server. For example, the request information could be stored on a backend database after communicating the application to a web server. Another example, the JavaScript might incorporate with Active Server Page (ASP) to apply validation modules on the server-side.

JavaScript has introduced several security problems including Denial-of-Service attacks, input validation attacks, script attacks, and privacy violations. For example, JavaScript methods can launch Denial-of-Service attacks on a client machine through a web page or electronic mail. A simple JavaScript Denial-of-Service is

to invoke `alert()` method inside a loop statement. Each time, the loop is executed a message window appears on the screen of a web browser. An adversary could construct a particular alert message to consume CPU resources. If a large number of these messages were sent, it could lead to Denial-of-Service (27; 80).

In addition, JavaScript has indirect access to operating system resources and user data through a set of JavaScript methods and objects such as the "history", "navigator", and "cookie". For example, "history" object allows a user to discover the URL of all of the other web pages that have been visited during a session. This feature could be combined with the above feature to perform a form of automated eavesdropping attack (27; 80).

However, the HTML form content might be faked by cancelling the validation modules on the client-side. The server is fooled and then accepts the faked form. This is possible, because any user can display HTML and JavaScript code using web browser tools. Microsoft Internet Explorer can display the HTML source code for web pages but it cannot display encrypted scripts or compiled modules such as ActiveX controls (7; 67).

Safe-type programming language (such as Java) can explicitly declare the type of user inputs during program writing and then this gives less error and fewer security risks. JavaScript considers all user inputs have the same type and it is possible to apply any operation (such as mathematical addition or string concatenation) on any type of user input. Consequently, this leads to escape harmful meta-characters via user inputs. However, JavaScript form inputs are vulnerable to validation attacks (27; 30; 73; 84; 85).

2.3.6 Common Gateway Interface (CGI) Security

CGI is a communication protocol between a web page and a program that is executed on a web server (17). It permits the input data of HTML forms to be sent to a server, which runs a CGI script and passes this data to it through the standard input or standard query. The CGI program can then process the data, and return it in HTML form format through the standard output to a web

browser. CGI can be written in any language (such as JavaScript, C, C++ and Perl) and can be run on various types of operating systems such as Mac, NT, UNIX or any operation system that runs on a web server (13; 35; 54).

One advantage of CGI is that it is used to generate dynamic web pages in relation to user interaction. For example, the creation of a web search engine, which reads a search string from a user, searches a database of web pages, and returns HTML data listing the matching sites (13; 24; 27; 54).

CGI returns the output data as part of the URL, and then output data is encoded to binary format. The blanks are represented by "+", and the other special characters are represented as "%XX" where XX is the ASCII value of a character in hexadecimal format. A CONTENT_LENGTH environment variable determines the number of bytes to be read on standard input or standard output. However, data tampering could be taken place when data has been altered either on the client-side or the server-side (80). For example, an adversary might bypass the data validation modules in a client, alter data input, and then the CGI scripts are tampered with. A tampered CGI script can disrupt a server, and might be exploited to gain access to a back-end database (8; 27).

CGI scripts only run on a server, and hence provide more opportunities to abuse data on a server. Many web sites adopt double-checking of data validation modules to ensure the integrity of a web system (80). However, the generated dynamic web content is an issue because the double-checking validation scheme is not able to ensure the data integrity against tampering.

Another security risk is that the HTTP is sessionless (24; 26; 27; 28). HTTP only provides once the type of request/response communication. Therefore, each time, there is a new connection between a web browser and a web server. CGI supports the maintenance of state through the use of hidden variables or cookies that keep track of the current information for each request (28; 30). However, this possible solution may be invalid through saving the HTML form to a disk, tampering the hidden values, and then reloading this altered form into a web browser for rendering (28; 31; 32).

2.3.7 Web Security Technologies

We have described the two basic technologies of web security: hashing and Message Authentication Code (MAC).

2.3.7.1 Hashing

Hashing is a web security technology that might assist to ensure the integrity of data. The hash function generates computationally unique hash values similar to fingerprint signature. The hash value is called a message-digest or a checksum that is expressed in hexadecimal or binary format (86).

The form of hash function is $h = H(p)$ where h is the checksum in binary or hexadecimal form, and p is the variable length of a web document. The checksum is compared to the previous calculated hash value of a web document for checking the web document integrity (27; 86). The hash function is an effective cryptographic hash method for the following features (27; 86; 87):

- One-way hash function: the adversary cannot decrypt the checksum because it is one way function.
- Random values: it gives different checksums even though the binary string inputs are similar.
- No collision: it computationally finds the values of two binary strings x and y in different locations, even though the $h(x)$ and $h(y)$ are similar.

Table 2.2 shows a comparison between a set of hash functions (27; 86). Although a variety of hash functions are available, only MD5 and SHA-1 are in wide use (88; 89).

Table 2.2: A list of hash functions

Hash Function	Hash sum size (bits)
MD4	128
MD5	128
RIPEMD-128	128
RIPEMD-160	160
SHA-1	160
SHA-256	256
SHA-384	384
SHA-512	512
Tiger / Tiger2	192

2.3.7.2 Message Authentication Code (MAC)

MAC is a very small code to ensure the integrity of data beside the cryptography techniques. It is a one-way hash function that involves a private key (86; 90). Only the parties that know the private key can compute the MAC value (90). Therefore, a private key can be produced by one of the parties, and then sent in an encrypted format to other using the public key encryption method (86). A sender computationally generates a number (fixed-length data item) that is formulated from a combination of the key and the message. On the other hand, a receiver uses the same key with a computational procedure to re-compute this number, and this is called authentication code. If the matching is true then the message is not altered (86; 91).

2.4 Conclusion

This chapter has discussed the data flow over the HTTP request-response model. Data integrity relies on the integrity of the HTTP request-response model, if this model fails, then the data integrity may be violated.

We have explained that the strengths, and limitations of the current web technologies (HTML/XHTML, Java, JavaScript, and CGI) referring to a number of reasons: the bad implementation of a web system might expose some restricted functionality of the native operation system, and the limitations of web servers and browsers such as the transparency of a code on a web browser.

We have offered a review of web security issues and the related problems at the different levels of a web system including the main supporting web technologies such as Java, JavaScript and CGI. For example, we have identified the vulnerabilities of JavaScript validation modules. There are three basic security issues: confidentiality, availability, and integrity of data. In this thesis, we have focused on one of these issues, data integrity, because it has received little attention in the information security research.

In the next chapter, we will look in more detail on the integrity of web content by identifying the tampering attacks on a server-side static and dynamic web content. We will also discuss the limitations, requirements, strengths and weaknesses of existing systems and approaches such as integrity verification systems and web engineering approach. In addition, we will propose our solution to tackle the research problem.

Chapter 3

Integrity Verification Systems and Related Work

3.1 Introduction

The recent advances in textual and multimedia technologies have made the manipulation of web content (such as text, images, videos or audio objects) very easy (21; 41; 42). The broad availability of these new capabilities have enabled numerous new applications. Web content can easily be tampered with by almost anyone. Therefore, investigating web content survivability (i.e. detection and recovery) and the control of the exchanged data have become a major issue. To counteract this risk, a number of traditional security technologies have been proposed to ensure the integrity of web content but they fail to provide any provable security guarantee against malicious modification attempts (41).

In the previous chapter, we outlined a general background of web content, HTTP request-response model, and web security, and we have tried to place web security in perspective. Although many of the verification and security mechanisms have been adopted to ensure web content over HTTP request-response model, a number of web security issues are still unresolved. Security problems are discovered on browsers and servers, while important security policies and technological questions have yet to be answered in a meaningful way (21; 41).

In this chapter, issues of server-side data tampering will be described in Section 3.2. Section 3.3 will define the survivability elements in Internet security research. We review the problems associated with unauthorised data manipulation of static and dynamic web content in Section 3.4. We also survey the existing integrity verification systems and other approaches such as the web engineering security approach in Section 3.5. The main finding in this chapter is that unauthorised tampering is still a potential problem because dynamic web content is not verified and not recovered. Therefore, we will propose a systematic web security system (framework and model) that could ensure the survivability of web content to internal and external users, even though a web data manipulation problem may have occurred in Section ??.

3.2 Data Tampering

In this thesis, we have focused on the server-side security because the server is the central system and the repository of data resources (21). Regardless of which web server application is running, many issues are common, such as web server configuration, access issues and of data integrity.

A poor web server configuration can lead to disaster. Many web sites are installed in the `cgi-bin` scripts and accept the standard configuration of a web server application without question. A web server has two roots: the server root which includes the control information and the document root which contains content and web resources of web sites and web applications (21; 54).

Web servers must deal with some type of active content such as CGI. As discussed in the previous chapter CGI scripts are usually written in Perl, Java, or C (21). However, CGI scripts run on a server and hence provide more opportunities to abuse data (80), as shown in Section 2.3.6.

For example, a criminal might also bypass the data validation modules in a client, alter data input, and then the CGI scripts are altered (see Section 3.4.3). An altered CGI script can disrupt a server, and might be exploited to gain access to a backend database (8; 27).

In our proposed web environment, an integrity objective is clearly defined as ensuring static and dynamic web content has not been tampered with (i.e. illegitimate modifications). Modifications include deletion or alteration of existing web content and each addition of new data to existing web content.

One of the issue of security exposure is the server-side data tampering (8; 27; 80). Some results of data tampering are:

- Modification or destruction of data on the server.
- Data theft which is an extreme to end of data tampering.
- A web server application integrity is compromised.
- A client integrity is compromised.

One suggested way to eliminate data tampering is tamper resistance. Tamper resistance is the manner by which normal users of a product, package, or system (or others with physical access to it) are prevented from tampering with the product, packages or system. There are a number of ways for employing tamper-resistance such as a secure cryptoprocessors, the IBM 4758 and chips used in smart cards (15). However, there is no one solution can be considered as "tamper proof". Often multiple levels of security need to be addressed to reduce the risk of data tampering such as (92):

- Identify who a potential tamperer might be.
- Reduce the risk of data tampering by using the appropriate knowledge, materials, tools and others.
- Describe all suggested feasible methods of unauthorised access into a product, package, or system. In addition to the primary means of entry, also consider secondary or "back door" methods.
- Enhance the tamper resistance to make tampering more difficult, and time-consuming.

- Add tamper-evident features to help indicate the existence of data tampering.
- Educate people to watch and monitor for evidence of tampering.

The tampering of dynamic web content is a real-time issue because the generation of dynamic web content depends on user interaction (32; 91). Different user information leads to different generated web content. Therefore, it is very difficult to analyse automatically and even manually the requested page of dynamic code before processing on a web server. The dynamic code of server programming languages needs to be processed on a web server before returning the response to a web browser. As a result, we cannot guarantee that dynamic code is not tampered with even if the static code is verified; therefore the generated web content should also be verified. Currently the dynamic web content can be tampered with using malicious web manipulation software (14). This software listens, monitors, and analyses the generated server-side web content. Therefore, the verification of dynamic web content is a challenge because the web developers, users, and others cannot predict the output of response.

The integrity of a web server environment depends on dynamic, unstructured data that is generated by running server-side scripting dynamic web pages (15). The key issue in this thesis is that even if the application knows that this data can impact on application integrity, the hashing measurement of static web contents on the repositories of a web server is useless because we cannot predict values that would preserve integrity. In addition, even though the server-side scripting dynamic web pages are verified, the integrity of generated dynamic web content can be tampered with.

In a web server, the key dynamic data are (15):

1. The various types of requests from remote clients, administrators, and other Servlets and
2. DBMS tables.

3.3 Survivability

The definitions of survivability have been stated by a number of researchers in (93; 94; 95). They have defined survivability as the capability of an entity to continue its mission even in the presence of damage to the entity. The damage caused by web attacks, system failures, or accidents, and whether a system can recover from this damage, determines the survivability characteristics of a system. A survivability strategy or scheme might be constructed from three elements: protection, detection, and recovery (95).

In this thesis, the high level strategy underlying for the proposed approach to designing a survivable system is to strategically combine elements of detection and adaptive reaction of in the architecture of the system. Individually, each element is not sufficient to secure the system from any possible damage, such that:

1. Detection provides awareness to the status of the system and allows the system to detect attack.

2. Adaptive reaction enables the system to cope with undesirable modifications caused by adversaries through supporting recovery.

Therefore, the need to combine the elements of survivability (i.e. detection, and adaptive reaction) is based on the understanding that protection cannot be fully sufficient, some attacks will succeed or partially succeed, and some of the attacks will not be detected in time (41).

In this thesis, we have assumed that all manipulation components are susceptible to malicious attacks. These attacks may involve tampering with the existing source code or content of referenced objects to include undesired functionality, or replacing a genuine web content with a malicious one. When using such components in the web system, or illegitimately installed by malicious scripts, it must check to see if a web content has not been modified in an unauthorised manner since it was created. Therefore, we have proposed a novel approach to satisfy the requirements by checking the integrity of web content.

3.4 Integrity of Web Documents

Below, a number of basic protocols and schemes are reviewed in order to give a more detailed overview of web document integrity. There are four established methods in common usage for ensuring the integrity of web content: the SSL protocol to secure the communication channel, digital signatures to provide security by verifying the authenticity of the web document, form-field validation modules to validate user inputs against harmful data at the client and server-sides, and firewalls to protect against malicious codes and other attack strategies such as IP spoofing and DNS spoofing.

3.4.1 SSL

There are three basic levels of communication channel. Each one consists of several protocols as follows:

1. Network Layer: contains the basic protocols such as IPv4, IPv6, and IPSec (96). The standard protocol is IP. These protocols are used to provide the integrity of data at the network layer of Open Systems Interconnections (OSI) model (96). They control packets by routing them to the correct destination. However, CERT (65) outlines that the IP-based network is vulnerable to threats such as IP spoofing.

2. Transport Layer: contains the SSL, and Transport Layer Security (TLS) protocols (23). SSL and TLS are used to secure the transit over a communication channel against eavesdropping and tampering and to authenticate a web server (13; 96). They are designed to support the authenticity and non-repudiation (i.e. ensuring that neither the sender nor the receiver are able to deny the transmission (67)) for communications through TCP/IP connections (97; 98; 99). However, authentication techniques (user identification and password) or public key certificates are not sufficient to protect web sites against web spoofing. SSL and TLS do not provide an absolute solution although they are cryptographic security protocols (96).

Readers should note that this thesis does not deal with integrity of data in transit. We deal with the integrity of data stored on web servers and the generation of dynamic web content during the HTTP request end response mode.

3. Application Layer: contains protocols such as HTTP, and SHTTP (44). These protocols specify some tasks such as security policies, cryptographic algorithms over the communication channel, and negotiation of key management mechanisms (13).

SSL protocol is located between the transport layer protocol and the application layer protocol and composed of two layers: the SSL record layer to provide secure connections and the SSL handshake layer to allow client-server authentication (97; 98; 99).

SSL controls the communication channel between a client and a server. Each server should have an SSL server certificate such as X.509 v3 digital certificate. A user interacts via a web browser to make a request to a web server. The web server sends its public key and digital certificate to a web browser. The aim is to authenticate the identity of a server and to encrypt the user data using a public key. Subsequently, a web browser checks the validity period field on the digital certificate to make a correct connection (86).

The main problem of SSL is that the security is only effective from a single point to another (100). Guest (100) outlines SSL is no longer applicable when the data reaches to end point over the SSL communication channel. For example, if we send a message from Client C to Server S, we can use SSL to secure the link between of the two. But if we send a message from Client C to another Server SS and it has to go via Server S – we cannot guarantee message security using SSL through this intermediary. Therefore, SSL is only designed to provide point-to-point security (6).

SSL has a number of limitations including (6):

1. SSL cannot help applications prevent information leakage and impersonation at an application gateway (such as proxy, web server application, web

browser, antivirus, interceptor and others) when an application gateway is required.

2. It only supports one type of cipher suit inside a connection.

3. It only supports one simplex channel, in which data is transmitted only in one direction.

4. It does not have sufficient negotiation mechanisms to enhance and provide end-to-end security.

A more complicated situation occurs when one or more intermediaries are present in the request-response chain. There are three common forms of intermediary: proxy, gateway, and tunnel (44). Figure 3.1 shows a finite number of intermediaries ($G_1 \ldots G_n$) between the Client C and Server S. A request or response message that travels the whole chain will pass through `n` intermediaries separate connections.

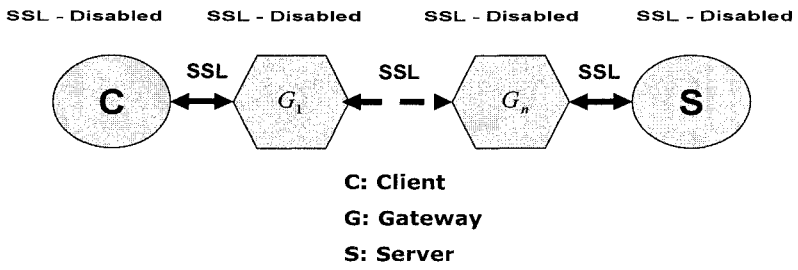


Figure 3.1: The end-to-end security model and gateway chain model of SSL

The end-to-end security of SSL is shown in Figure 3.1, in which there is a number of application gateways between Client(C) and and Server(S). In this model, SSL can protect the communication between any two neighbouring entities but cannot prevent a gateway in the chain from reading and modifying sensitive data (6). Therefore, these application gateways are able to tamper with the static and dynamic web content because the encrypted data could be exposed (decrypted) for processing either in a web server application or a web browser. As

can be seen in Figure 3.1 while SSL can provide a secure point-to-point channel, it has problems in the presence of application gateways: if an application gateway G is involved, client C normally sets up an SSL connection with G, and G acts as the delegate of C and sets up another SSL connection with server S (6).

3.4.2 Digital Signatures

Signature schemes normally consist of two algorithms including: a signing algorithm which requires the private key and another for verifying signatures which requires the public key of user. As a result, the output of the signature process is a digital signature (7; 12; 86). Digital signatures can provide security by verifying the authenticity of a document. However, a criminal may replace the current web page by an expired original web page through circumvention of digital signature (7; 12). Therefore, it is necessary to indicate when the document was signed because if the private key of the signer was lost or compromised, we cannot trust this signed document. This type of attack cannot be detected even if the web browser has enabled SSL digital certificates (101). To avoid this problem, the time-stamp signature should be provided. This time-stamp signature provides relative temporal authentication that every document is related to a relative time-stamp which positions the document at a particular point in time relative to documents stamped before and after it.

Time-stamp signature help to detect two types of forgery (7; 12; 101):

1. backward forgery (i.e. stamping a "past" time-stamp on the "present" document), and
2. forward forgery (i.e. stamping a "present" time-stamp on a "past" document or even "future" time-stamp on a "present" document by an unauthorized adversary).

However, the verification computation is proportional to the number of the issued time-stamps where the producing a time-stamp signature computationally is more demanding than producing a digital signature and this will increase the

server overhead. ***Readers should note that even though the time-stamp signature can secure the originality of a document it is impractical to secure dynamic web content such as the shares and prices because dynamic data involves higher calculation costs than static data*** (7; 12; 101).

We suggest using the hash functions in our proposed approach rather than the digital signature because the hash functions are faster by 1000 times than the RSA algorithm and they are 100 times faster than the RSA signature verification system (102).

3.4.3 Form Field Validation Scheme

Currently, when initiating user and organisation transactions and conducting their business, e-Commerce applications rely on HTML forms including enrolment, authentication, order entry, payment, and profiling, rather than XML forms, because of a lack of security mechanisms supported by SOAP (29). In addition, the XML form is not supported by the major web browsers because it needs additional installations on the client and server-sides (29; 52). Furthermore, the Gartner group has reported that the HTML events such as mouse and keyboard actions in the HTML/XHTML forms have become very widely used in business applications during 2004 through 2008 (25).

A form field validation scheme is the first defence against data tampering at the application level. Web developers have adopted a number of validation approaches to prevent loss of web content integrity:

- Server-side validation: this approach can be used to validate sensitive data on a server before processing them by an application server. Depending upon the application and network traffic, the time taken between the submitted form on a web browser and the error message that is returned from a web server can be considerable. In addition, it makes excess network traffic to enter the correct data format. Furthermore, it needs to define explicit server-side programming languages (52; 103).

- Client-side validation: this is effective for minimizing the number of necessary communication hits between the submitted form and received error message (52; 104). However, the form validation modules of this approach can be cancelled or removed. In addition, this approach cannot ensure that the client and server are authentic. Brabrand and others (50; 52) suggest validating the field input on a client-side before clicking on the submit button for more efficiency.

- The double-checking validation: this approach duplicates the form validation modules on both client and server sides. This approach adopts alternative validation scheme on a server-side, even though the validation scheme is bypassed at the client-side. However, this approach is expensive and involves high latency (52; 104).

- Honkala and Vuorimaa (29; 59) propose extending the XForm form to a digital signature XForm. They adopt the digital signature for XForm forms rather than HTML forms because it is hard to apply a digital signature to an HTML form. They advocate the "what you see is what you sign" approach to secure web form components at the client-side. Therefore, XForms is a new standard for better graphical interfaces and specified to input validation rather than the embedded scripts (103). However, XForm is only supported by the `<XSmile>` browser.

- Brabrand and others (50; 52) introduce a PowerForms form for input validation. PowerForms form is implemented by the XML language to define a rich form that includes input validations without using a scripting language. PowerForms is only supported by the `<bigwig>` browser. However, unlike Firefox and IE, this browser is not free. In addition, it requires additional installations to interpret PowerForms forms on a web server because the `MAWAL` language is used in the `<bigwig>` system for domain of interactive web services. This language is designed to protect the client sessions as extended to SSL technology.

- Formatta organisation defines a Portable Form Files (PFF) form model that is not related to the HTML language. This form allows the user to

encrypt and lock its form data before submitting it to a web server. However, the PFF form needs special software to install a web document on a web browser. In addition, the submitted data is sent by E-mail service to an organisation web server (7). Full details have been described in <www.formatta.com/>.

Many online book re-sales (such as Amazon) advocate checking inputs with JavaScript as a mechanism to reduce network traffic; modern books applications usually advocate doing input validation on the server for security purposes. Nevertheless, major e-Commerce and e-Service sites still use client-side validation and hidden fields (105). Authors (105) found client-side validation on amazon.com and netflix.com, and the use of hidden form fields to store sensitive information on fastlane.nsf.com.

A validation scheme is necessary for both client and server-sides, but is not sufficient to protect web content integrity against tampering, because fundamentally a form validation scheme is designed to validate basic properties of the input data: length, range, format, default value, and type. In addition, data validation can be used to enhance resistance to injection attacks such as SQL injection attack because SQL injection vulnerabilities result from insufficient data (input) validation (106). However, a validation scheme is useless if any malicious script or listener is already installed on a server (4; 32; 52).

Furthermore, web developers assume if the validation modules are properly managed then the static and dynamic web content will be secured. Unfortunately the attacks could come from inside the orgranisation and this approach cannot validate the data on the fly. HTTP provides the REFERER header to help in the detection of a tampered form. A REFERER contains the URL of the original form. However, this header is not sufficient to alert a web server and web browser because it is possible to tamper with the URL information of the header (7; 44). As a result of the transparency of code at the web browser level, the following approaches can cause loss of content integrity at the HTML form level:

- Hidden fields manipulation: an adversary saves the HTML form to a disk,

modifies a hidden field value (such as the price of a product), and then reloads this tampered form into a web browser for rendering (32).

- Script manipulation: an adversary removes the client validation modules from a web browser to submit illegal data to a web server. A web server accepts the tampered form and then the data is saved in a backend database. Many web application security vulnerabilities come from input validation problems including Cross-Site Scripting (XSS) and SQL injection (4; 7; 12; 73). This approach is made possible by removing all script modules between the `<script>` and `</script>` tags, removing the event-handler that invokes the validation modules, or turning off the script and Java Applet options via web browser settings.

- Modules of validation analysis manipulation: an adversary applies reverse engineering techniques on the validation modules (4; 7; 73).

- Session information manipulation (67): an adversary might access session information, which is saved in cookies. This is possible because a web browser is not fully secured by SSL. In addition, some client-side programming languages provide direct or indirect commands to access user machine resources. However, the cookie-securing mechanism of Park and Sandhu (28) can be adopted to avoid or reduce the danger of tampering the cookie information but it needs to declare the explicit modification to the existing web environment. This mechanism is implemented using CGI and Pretty Good Privacy (PGP) to verify secure cookie cryptography on the client. The encryption scheme is conducted by the server. Therefore, the cookie-securing mechanism is used to secure the session information on the cookies for continuing the Request-Response conversations.

3.4.4 Network and Application Firewalls

Network firewalls provide protection at the host and network level (8; 31; 32). There are, however, four reasons why these security defences cannot be used to detect data tampering attacks (8; 37):

1. They cannot stop malicious attacks that perform illegal transactions, because they are designed to prevent vulnerabilities of signatures and specific ports.

2. They cannot manipulate form operations such as asking the user to submit certain information or validate false data because they cannot distinguish between the original request-response conversation and the tampered conversation.

3. They do not track conversations and do not secure the session information. For example, they cannot track when session information in cookies is exchanged over an HTTP request-response model.

4. They provide no protection against web application attacks since these are launched on port 80 (default for web sites) which has to remain open to allow normal operations of the business.

There are some general approaches that are adopted to protect the web environment in most commercial security products such as application and network firewalls (72). These are detailed below:

- Analysis Approach: analyse code and reject it if it could harm a web environment. Analysis approaches include scanner techniques that scan a file and reject it if contains any known virus and compilers that can determine previously known malicious code or detect the code bugs.

- Rewriting Approach: rewrite code before running it. This includes a rewriting tool that adds extra code to perform dynamic checks that ensure bad things such as destructions or disruption of data do not occur. For example, a Java compiler adds extra code to check that every array index is in bounds - otherwise, the code throws an exception, thereby avoiding the buffer overrun attacks.

- Monitoring Approach: monitor code during the execution and catch it before it does anything bad. Monitoring approaches include a reference monitor which is used to ensure programs do not do anything bad. For example,

an operating system uses the page-translation hardware to monitor the set of addresses that an application attempts to read, write, or execute. So that, if the application attempts to access outside of its address range then the kernel makes an action (e.g. by signaling a segmentation fault.).

- Auditing Approach: audit code during execution. Commercial auditing tools create an audit trail that captures program behavior.

Firewalls are necessary, but they are not sufficient to ensure web content integrity against tampering because fundamentally they are designed to provide protection at the host and network levels. Therefore, they are useless if any malicious script or listener is already installed on a server behind them because they do not prevent the installation of scripts at the application level.

For example, once inside the network and running on a user's machine, the firewall offers no protection against the mobile code accessing sensitive internal data and may be leaking it to outside of organisation. The security must happen at the user's machine rather than at the firewall. Previously, a firewall could suppose that an adversary could only be on the outside. Currently, with mobile environment, an attack might originate from the inside as well, where a firewall can offer no protection (107).

3.5 Related Work

In this section, we present the related work in the area of integrity verification systems and the area of web engineering security. Below, a number of existing approaches and systems have been reviewed in order to give a more detailed overview of integrity verification. In this thesis, we have surveyed existing approaches, considered their strengths, weakness, and limitations. Some of these critical solutions are summarised in Table 3.1, which shows the source of the problem (client, server), the position of the solution (client, server) and the modifications necessary on the client and server. We start with web engineering approach.

Client-side Encryption System	Dynamic Security Surveillance Agent (DSSA) System	Adaptive Intrusion-Tolerant Server System
Solution: Client **Modification:** (Smart Card) **Type of web content:** User input +Session information	**Solution:** Server **Modification:** Server **Type of web content:** Static web content on the server.	**Solution:** Server **Modification:** Server **Type of web content:** Static web content on the server.
Technique: o Cryptography scheme on the Client-side. o Use smart card for generating the private key. o Use hash function.	**Technique:** o This system uses a timestamp signature and hash function. o If DSSA detects any modification, it signals the Web server to block their transmission and take the preventive action.	**Technique:** o This system comprises redundant servers running on diverse operating systems, intrusion-tolerant proxies and client machines to verify the behaviour of servers and other proxies, and monitoring and alert management components. o Use hash function.
Drawbacks: o Generated key might be violated or lost. o JavaScript that calls applet functions and Java Applet can be cancelled or tampered with. o It fails to assess an existing web application. o Tampering is still potential problem because this system does not verify the integrity of dynamic and static web content on the server.	**Drawbacks:** o Tampering is still potential problem because DSSA does not verify the integrity of dynamic web server content.	**Drawbacks:** o Tampering is still potential problem because the adaptive intrusion-tolerant server system does not verify the integrity of dynamic web server content. o Performance is too slow if more than three application servers are needed.

Table 3.1: This table shows the current solutions, the source of the problem (client, server), the position of the solution (client, server) and the modifications necessary on the client and server.

3.5.1 Web Engineering Security Approach

Glisson et al. (108) suggest that security should be visible in all steps of the development process if it is to be implemented with any success. In other words, web engineering principles, such as design, implementation and testing, should be integrated into web security to identify what a user and an organisation need for every stage of the software engineering principles. Therefore, Glisson and Welland (17) propose a Web Engineering Security (WES) methodology. The WES methodology serves to detect the vulnerabilities at each stage instead of processing them at the implementation stage. They conclude that technical tools alone will not solve current security issues and so it will be effective to incorporate security component upfront into the development methodology of web engineering phases. Glisson and Welland (17) take the advantage of existing security tools within the organisation. However, this integration may require the rebuilding of existing web applications as some consist of complex structures, comprising multiple programming languages and imported binary components with little or no documentation. Consequently, it may be difficult to define security vulnerabilities for legacy web applications at every stage of the software engineering life-cycle all the web policies that maintain security vulnerabilities.

Success comes to organisations that build security into all phases of their application development lifecycle (109). Vulnerabilities typically find their way into applications during each major phase of development – requirements collection, application design and application implementation. Every development team's goal should be to identify all relevant security requirements at the same time that functional and performance requirements are collected. Once good security requirements are in place, the team should strive to identify vulnerabilities during the design, rather than discovering issues during implementation and going back to re-design pieces of the application. Therefore, security practices should be in place during requirements planning, design, implementation and testing in order to catch the majority of problems as early in the cycle as possible. Proper training on how to integrate security into each of these phases will provide consistency across the organisation. It is less expensive and less disruptive to discover

design-level vulnerabilities during the design rather than discovering them during implementation or testing (109).

3.5.2 Integrity Verification Systems and Approaches

3.5.2.1 Client-side Encryption Approach

Hassinen and Mussalo (9) have proposed a client-side encryption system to protect confidentiality, data integrity, and user trust. They encrypt data inputs using a client encryption key before submitting the content of an HTML Form. The client encryption key is stored on the server and transferred over an HTTP connection. A downloaded web page includes a signed Applet which handles the encryption and decryption values. This applet also reads the encryption key from a local file. In addition, a downloaded web page includes JavaScript methods that invoke the Applet methods for encryption and decryption. This approach uses a one-way hash function. It computes the hash value which is inserted into the main data input before encryption. After a new request, the JavaScript function invokes the Applet decryption method to decrypt the parameter value and places the returned value in the corresponding input field. The message validation includes finding a new hash value from the decrypted message and comparing it to the hash value which is received with the message. If they are the same, the data is accepted, otherwise, the data is deemed to have been altered and the validation will fail. This system has two main requirements (9):

1. It must be able to run on any major web browser.

2. Without the need to install additional hardware or software.

However, data integrity could be lost if this approach is adopted because the Java Applets can access the client's local file system. Thus, a criminal can replace the original signed Applet with a faked Applet to access the client's web content. Moreover, if a smart card is used to store the client encryption key, then the client machine requires a card reader and necessary drivers, which fails the second of the

above requirements. This encryption system can use the Java Card technology for two aims:

1. storing the generated key on the card so that it can be read only with a PIN to control authorization of the user.

2. storing the Applets that executed on the card.

Another potential weakness is the potential loss of the client's smart card with its Personal Identification Number (PIN), in which case the whole web-based system would be compromised. In addition, the Applet and JavaScript methods can be bypassed. If this happens, the submitted values will be in plain text. Furthermore, in order to implement this technique, existing web applications will require modification. Finally, it does not include a risk analyser to protect user information on the web server if the web server has been tampered with.

3.5.2.2 Dynamic Security Surveillance Agent (DSSA) System

Sedaghat, Pieprzyk, and Vossough (7; 12) have proposed a Dynamic Security Surveillance Agent (DSSA) system on the server-side that automatically intercepts an HTTP request to verify the integrity of the requested page before the web server responds to the client. This system is positioned between the web server and the client's machine where the DSSA uses a timestamp signature and hash function to verify the integrity of the full content of a requested web page including all of its embedded objects. If the hash value of the requested web page and its referenced objects equals the original hash value, which is registered in a secure database, the web server accepts the HTTP request and can respond to the client. If DSSA detects any modification, it signals the web server to block their transmission and take the preventive action such as DSSA sends an email message to alert the web site administrator. At the same time, DSSA attempts to send the requested page from a secure server to the user. In this situation, DSSA automatically rejects any further communication from the clients to the Web server until the problem is resolved.

However, as the DSSA does not verify dynamic web content which is generated on the fly, tampering is still a potential problem. Therefore, verificition of dynamic web pages are still a significant challenge. Furthermore, DSSA considers a tree scheme of digital signatures – which provides that a different message (hash value) be signed for each node (such as a web page) in a tree. If two or more nodes (such as web pages) reference the same object (such as an audio object) in a node, then the signature of that web page will be different from other web page. It assumes that every web page is independent from every other web page, even though some referenced objects are shared by more than one web page. Therefore, the performance is decreased during the verification of a web page.

Furthermore, DSSA supports a recovery component to recover the whole web page with its referenced objects, so that if any referenced object has been tampered with the DSSA recovers the whole web page and its referenced objects. Consequently, the system overhead is relatively high.

Readers should note that the notation of dynamism in this system does not apply to generation of web content.

3.5.2.3 Adaptive Intrusion-Tolerant Server System

Valdes et al. (110) propose an adaptive intrusion-tolerant server system for fully static web server content. The adaptive intrusion-tolerant server system (110) comprises redundant servers running on diverse operating systems, intrusion-tolerant proxies that position between web servers and client machines and verify the behaviour of servers and other proxies, and monitoring and alert management components based on the EMERALD (111) intrusion-detection system. However, the system adapts its configuration dynamically in response to intrusions or other faults. The adaptive intrusion-tolerant server system does not employ fault-tolerant replication protocols, but has a range of adaptive responses to isolate compromised parts of the system. This approach addresses the data integrity and data availability on a server: the system must be capable of servicing requests from correct clients even though one component of architecture is compromised.

When a client request arrives, the main proxy (leader) accepts client requests, forwards them to a number of application servers, and compares the hash values of the content returned by the application servers. If they match, the main proxy (leader) sends a response to the user, otherwise, a report is sent to a monitoring component for the correct action to be taken. The main proxy and application servers communicate over a private network that is monitored by an Intrusion Detection System (IDS). The IDS provides detection for any compromised application servers, so that compromises are likely to remain limited to a small number of application servers. An agreement policy determines which and how many application servers are involved by the main proxy for each client request. Therefore, a single client request will be served by one, two, or more web servers.

Valdes et al. (110) have identified three assumptions to design the architecture of adaptive intrusion-tolerant server system as follows:

1. The main proxy (leader) implements an agreement policy that serves correct content.
2. The leader is secure.
3. Only one critical component (such as application server and monitoring proxy) is compromised at the same time if intrusions occur.

However, because the adaptive intrusion-tolerant server system does not verify the integrity of dynamic web server content that is generated on the fly tampering remains a potential problem. Moreover, it cannot detect tampering attacks such as visual spoofing attack on static web content. It only tends to detect security vulnerabilities such as worms (e.g. code red virus). In addition, it does not support recovery if all the application servers are compromised. As stated in (110), the updates or the new modifications must be done offline.

Furthermore, the performance is too slow if more than three application servers are needed to check the integrity of web server content where this system requires redundant diverse web servers and various operating systems on diverse platforms to be able to provide a greater assurance of availability and integrity.

3.5.2.4 Application-Level Gateway Approach

Scott and Sharp (32; 112; 113) propose a gateway model which is an application-level firewall on a server for checking invalid user inputs and detecting malicious script (e.g. SQL injection attack and cross-site scripting attack). They have developed a security policy description language (SPDL) based on XML to describe a set of validation constraints and transformation rules. This language is translated into code by a policy compiler, which is sent to a security gateway on a server. The gateway analyses the request and augments it with a Message Authentication Code (MAC).

The MAC is used to protect the data integrity. For example, The MAC is used to secure session information in a cookie at the client-side. There are many ways to compute a MAC such as one-way hash algorithms (MD5, SHA-1) to create an unique fingerprint for data within the cookie. Further details regarding MACs are described in (86).

However, this approach has a number of limitations. One of these is that tampering is still a potential problem because dynamic web content that is generated on fly is not verified. Therefore, if referenced objects are tampered with the gateway cannot alert the web administrator and clients. Furthermore, the policies of validation constraints and transformation rules are enforced manually and need an engineer to write and check them by hand.

Further limitations of this approach are that it enables protection through the enforcement of strictly defined security polices but does not address tampering problems for referenced objects. In addition, it is difficult to define all the policies and rules of a legacy web application for every single data entry point, URL entry, and cookie unit because these can consist of complex structures of multiple programming languages and imported binary components with little or no documentation. Therefore, it is not practical for a web administrator or publisher to be familiar with all of the data entry points for an existing web application that contains many static and dynamic web pages. Finally, it is difficult to adopt this approach where web applications are implemented via a large number of web

technologies because the policies and transformation rules are enforced manually, and need an engineer to write and check them by hand.

3.5.3 Section Summary and Conclusion

Because of the statelessness of HTTP and the limitations of SSL, data integrity can be violated. Static and dynamic web server content can be tampered with through running malicious code behind the firewalls on the server. This code can compromise a requested page before the client receives it (4; 7; 8; 9; 11; 14). Ensuring static and dynamic web content integrity against unauthorised tampering, even when the communication channel between client and server is protected has therefore become a major challenge. The question this thesis addresses is how the integrity of server-side static and dynamic web content can be verified before the client receives the requested page and then to provide continued reliable and correct services to users, even though a tampering problem has occurred. Indeed, this chapter has reviewed the problems associated with unauthorised data manipulation of static and dynamic web content and the existing approaches, considered their strengths, weakness, and limitations.

We conclude that the above existing solutions are not sufficient to tackle our research question. For example, SSL is a cryptographic protocol to secure the data in transit against eavesdropping problems. Therefore, the ends of web-based system (such as web browser and web server) remain unprotected. The user information, source code of web pages, code of web applications, and reference objects remain unprotected against tampering problems.

We also suggest using the form validation modules. They can validate the user information from harmful characters that cause input attacks such as XSS and SQL injection. We recommend that these modules be operated on the client and server because if the client validation modules is subverted, the server validation modules can still work. Some developers suppose that if user information has been properly validated, the static web content (source code of web pages, reference objects, code of web application) will be secure. However, malicious code might

be installed on a server either inside or outside organisation. Furthermore, as seen above, the form validation modules can be bypassed.

Network and application firewalls are necessary but they are not sufficient to protect the ends of web system. Firewalls can be used to protect against DNS spoofing attacks and penetration of ports at the network and host levels. Furthermore, firewalls do not understand the HTTP conversations between clients and servers. In this thesis, we have distinguished between the network level and application level because the security vulnerabilities of a network level is different from an application level.

Client-side encryption approach is used to protect user information. Even if the client validation modules is enhanced by an encryption approach, the static and dynamic web server content can be tampered with because moving the encryption to a client-side is vulnerable to security risks. The encryption can be exposed through applying penetration strategies such as reverse engineering techniques.

DSSA system and an adaptive intrusion-tolerant server system cannot ensure the survivability of server-side dynamic web content. However, even if the static web server content is verified, the dynamic server data can be tampered.

3.6 Conclusion

In this chapter, we have discussed data tampering and the definition of the survivability. In addition, we have reviewed the existing approaches and considered their strengths, weakness, and limitations. This thesis focuses on the integrity of data. We have shown, that given the limitations of the SSL protocol, a loss of data integrity is made possible by the statelessness of HTTP. In an attempt to overcome this problem, we propose the WCVR system that can provide continued reliable and correct services to internal and external users, even though a web data manipulation problem may have occurred.

In the previous chapters, we have discussed the statistics from CERT that have

shown a large year on year increase in reported security incidents and vulnerabilities. We have also shown that traditional approaches to security - specifically network firewalls and the SSL protocol - are insufficient to deal with tampering attacks arising due to the increased logical and physical accessability to organisations.

In the next chapter, we will describe the design of the proposed WCVR system to tackle the research question. In addition, we will compare the proposed solution with the existing approaches, systems and schemes.

Chapter 4

Design of WCVR System

4.1 Introduction

This chapter presents a novel system architecture to investigate the survivability of server-side static and dynamic web content against tampering attacks. Chapters 1-3 have surveyed the critical factors and challenges that underpin this thesis. A significant challenge to identify tampering problems is a difficult analysis of dynamic web pages (32; 91). Adversaries can evade server-side web content by using malicious web content manipulation software to produce malicious web content at the run-time (14). Chapter 2 and 3 have included a summary of the HTTP request-response model and how to follow the data flow over this model, web security issues, limitations of web technologies and an overview of related work and existing integrity verification systems and approaches.

To design a software system, several architecture styles have to be combined to construct the overall architecture of the system (114). The various architecture styles or architectural approaches relies on the fact of what kind of functional or non-functional quality requirements the system should satisfy. In this study, the satisfaction level of system requirements is evaluated through estimation the technical parts of architectural approach. Our system architecture relies on a number of work assumptions that derive from the literature review in chapters 2 and 3. The proposed framework architecture consists of five components: a web

register, a response hashing calculator, an integrity verifier, a recovery component and DBMS tables. Readers should note this thesis focuses on an integrity system and does not focus on developing or deploying a new hashing algorithm.

It should be noted that the current recommendation for newly designed applications is SHA-256 because SHA-1 may be broken (115). However, we have continued to use SHA-1, because:

- SHA1 has been tested in many applications and can be implemented using Java, C#, and C++; while the SHA-256 has not been fully tested yet (12; 74; 110).
- Although a variety of hash functions are available, only MD5 and SHA-1 are in wide use (88; 89).

4.2 Web Content Verification and Recovery (WCVR) System

As discussed in chapters 1-3, the problem to be addressed is that the integrity of the dynamic and static web content on a server can be compromised even though the communication channel between the client and server-sides is secured. Therefore, we have developed a novel system called the Web Content Verification and Recovery (WCVR) system for investigating survivability of server-side static and dynamic web content against tampering attacks before the client receives the requested resource. The WCVR system consists of a web security framework that executes a state protocol to enforce a set of web policies.

In this research work, a conceptual model called the Client Interaction Elements (CIE) model is formulated with the aim of understanding how to dynamically produce units of web content. Another important aim is to devise a simple method for automatically and manually analysing a web site that contains a large number of elements and where each web page includes a large number of interaction elements (e.g. forms, frames, links, and HTML and non-HTML objects).

As there is always a trade-off between security and performance (110; 116), we have then developed a new hashing strategy to utilise the hashing calculations for static and dynamic web content.

An evaluation of the proposed CIE model and new hashing strategy by the experimental work are presented in chapter 6.

4.2.1 Work Assumptions

Before we describe the components of our framework architecture, we have established four work assumptions because without such restrictions, there would always be adversaries (inside or outside organisation) that are able to fool remote clients and server.

1. The SSL protocol secures data in transit (7; 8; 9; 12); it provides digital certificates to encrypt the data in transit against malicious coding and eavesdropping attacks. Further details can be found in Section 3.4.1.

2. We assume that a data validation scheme should operate on both sides (client and server) to validate the user inputs of a request before it is processed on a web server. This means that if the client validation scheme is subverted, there is another validation scheme still running on the server before the request is processed. This is because the data resulting from user interaction on the client is inherently untrustworthy; user input may contain malicious code (such as SQL injection code) that could harm a web server or a backend database; or the client validation scheme can be violated (32). As a result, all user data that is sent as a request to a server is untrustworthy. Further details can be found in Section 3.4.3.

3. We also assume that the authentication scheme, authorisation scheme and Access Control List[2] (ACL) are correctly configured across a network (20; 21).

4. DataBase Management System (DBMS) is created, maintained and operated in a secure manner for ensuring the correctness of the internal state of the DBMS. Therefore, the protection of the hash value database is assumed.

4.2.2 Overview of Web Security Framework Architecture

An illustration of WCVR system architecture is presented in Figure 4.1. This framework consists of a number of components (117):

- DBMS tables: the DBMS contains two tables; offline-transaction table for mapping the hash values of static web contents to their specific repositories

[2]A list of all user accounts and groups that have been granted access for the file or directory on an NTFS partition or volume, as well as the type of access they have been granted. When a user attempts to gain access to a web resource, the ACL must contain an entry, called an access control entry(ACE), for the user account or group to which the user belongs. Further details can be found in http://www.iexbeta.com/wiki/index.php/Tech_Dictionary.

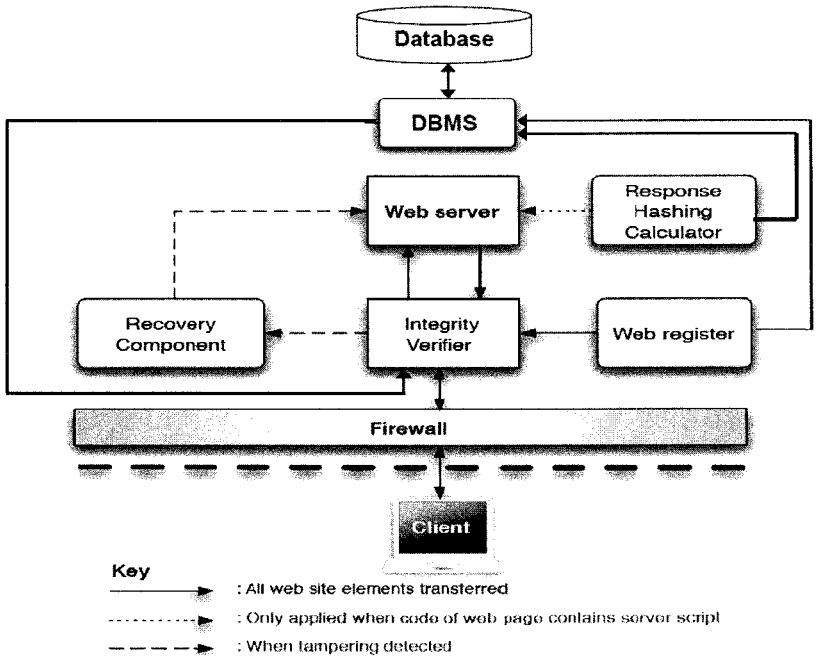


Figure 4.1: Schematic view of WCVR architecture

of web servers and online-transaction table for mapping the hash value of current dynamic web content to its server scripting web page. As well as the DBMS online-transaction table, any dynamic web content has be sent to a client, DBMS deletes the record of this web content because it contains only the current online transaction that is process.

The DBMS can maintain details about web contents in the background by rendering a specified relation as an online-transaction table, and offline-transaction table. In the DBMS offline-transaction table, an important property of all static web contents are stored is that it is append-only; modifications only add information with no information deleted. Hence, if old information is changed in any way then tampering has occurred.

The property of online-transaction table is append-edited; modifications add information and delete information.

The WCVR system supports DBMS tables to store and maintain every

detail about web content. Current approaches such as the client-side encryption approach (9) and application-level gateway approach (74) add the original hash value between the tags (such as `Meta` tag) of a HTML or XML response, and as a result, the malicious web content manipulation software could intercept and analyse the output response to obtain the original hash value so it is easy to evade the web system by tampering the original hash value of web content.

- Web register component: manages the hashing calculations for all static web content that have been developed for use in the secure environment. In this stage, the hashing calculations are done offline. If any static web content undergoes any add-on (new code that is added to the original web content (93)) , it is released. The updated web content must be regenerated by producing a new hash value. In addition, a web register sends the list of hashing values to DBMS offline-transaction table for storing and maintaining.

  It also monitors all resources of the web site and web application in their secure repositories for any authorised change that occurs. A repository constructs a list from static web content that have been developed for use.

- Integrity verifier (manager): mediates between the server and client machines by managing the HTTP requests and responses via a state protocol, which enforces a set of web policies that apply to the elements of the web system and to communicate between the other components of the WCVR system. The web policy specifies the next action or decision to be taken and when the policy should be enforced. In this thesis, we have defined four web polices, including request availability policy (policy 1), integrity failure policy (policy 2), integrity passing policy (policy 3) andrecovery policy (policy 4).

  When DBMS sends the hash value (checksum) of a request or response to the verification process, the verification process checks to see if the web content has been modified since it was used. The cryptographically original checksum is obtainable through the DBMS. The checksum of the requested

content is calculated and compared with the one retrieved from the DBMS table. Any tampering causes the content integrity check to fail. If there is a mismatch between the calculated checksum and the original checksum, the content integrity check will fail. Based on whether the test passes or fails, the integrity verifier component executes the state protocol to enforce the policy that makes the decision about the next step in the process. If the integrity check passes, the web content is sent to the running process straight away. If it fails, it is sent to the recovery component.

- Response hashing calculator: aims to do hashing calculations and backup for the generated dynamic web content before response sends it back to the manager.

- Recovery component: recovers the tampered web content if the action of the enforced web policy from the integrity verifier is satisfied. In other words, if the integrity verification process fails, it is sent to the recovery component, which tries to extract the original web content that is known to be safe. Once it is determined that the web content has been modified in an unauthorised manner, the system will try to recover the original web content through restoring it from the secure backup, put it in a new assembly and discard the tampered web content. When the new assembly is generated, the recovered assembly is sent to its direction and execution continues as normal.

Figure 4.1 illustrates how the proposed framework is separate from a web server. Note that the components of the framework do not need to run on a dedicated machine, they can be run as separate processes on the server.

It is suggested that the WCVR offers integrity of data, and a higher level of trustworthiness to organisations and the users. The proposed framework should be capable of verifying web pages and referenced objects in the designated directories of the web server and dynamic web content against tampering. To investigate survivability of web content, the WCVR system is able to detect web content against tampering and recover the original copy of the compromised object. Furthermore, at the monitoring stage, if a web system has been tampered

with, it provides alerts to the web server administrator. The WCVR system has a number of advantages over other approaches:

1. It does not require modifications to existing web application architectures.
2. It does not require any additional changes on the client and server.
3. It is compatible with all major web browsers.
4. It does not rely on a database of known tampering attacks.

It should be noted that we compute a SHA-1 hash with a changeable private key over the complete contents of the web resources. The resulting 160 bit hash value identifies the resource's contents. Different resource types, versions and extensions can be distinguished by their unique checksums.

Further, an essential part of our architecture is the ability of web administrator to ensure that the hashing list in a secure DBMS tables is:

- fresh and complete, i.e., includes all hash values up to the point in time when web contents are updated or modified legitimately.
- unchanged, i.e., the original hash values (original checksums) represent the original static and dynamic web content and have not been tampered with in a secure DBMS tables. Note that the operating system's web policy indicates that the access to this hash list is only permissable to web administrator or master of web server (refer to `assumptions` 3 and 4 in Section 4.2.1).

4.2.3 New Model of Interaction Elements

We have formulated a new model to deal with every interaction element (both HTML and non-HTML objects) in a web page called Client Interaction Elements (CIE) model. A CIE model extends Offutt et al.'s (4) HTML Input Units (IU) model which is only used to HTML input units to create tests on the client for web applications that violate checks on user inputs.

A CIE model is formulated to provide the automatic extraction of parameters (e.g. HTTP transfer mode, user, data, type of interaction element, and server page) of the client interaction element. The automatic extraction method provides a simple method for automatically analysing a web site that contains a large number of elements and where each web page includes a large number of interaction elements (e.g. forms, frames, links, and HTML and non-HTML objects).

Offutt et al.'s (4) also propose Input Validation Testing (IVT) that checks user inputs to ensure that they conform to the requirements of web applications. A common scheme in web applications is to perform input validation on the client-side with scripting languages such as JavaScript. Bypassing input validation modules on the client-side can cause faults in a web browser and can also break the security on web applications, leading to unauthorised access to data, system failures, invalid purchases and entry of malicious data. Therefore, the data integrity can be violated.

Note that, our Client Interaction Elements (CIE) model is used for server purposes whereas the HTML Input Units model is used for client purposes.

Ricca and Tonella (105) propose an analysis model and corresponding testing strategies for fully static web page analysis. Our proposed model can be used for static and dynamic web pages analysis. This is because the recent developments in web technologies and web applications are currently are being built on dynamic content, and therefore our model is proposed to deal with these static and dynamic web pages.

As discussed previously, web applications contain static web files and programs that dynamically generate HTML or XHTML pages (4; 7; 14). Each web page, whether a static web file or dynamically generated, may have either no interaction elements, one interaction element or more than one interaction elements that make users interact with a web server via a web browser. An interaction element I is characterized by five parameters: $I \cong (IT, SP, D, T, U)$. IT is the type of interaction element that can be used for interacting with a web server. The data inputs D are sent to a server page SP for processing user requests.

D (user input) is a set of ordered parameter-value pairs (p_i, v_i), where p_i is a parameter name, and v_i is the value of a data item (e.g. text box, selection list, etc.) that can be assigned to p_i. T is the HTTP transfer mode (viz. GET, POST, DELETE, PUT and HEAD). Each HTTP transfer mode T has a different method to transfer data to SP (44). U is a user who interacts via a web browser to conduct HTTP conversations. A user U can legitimately access web content on designated directories of a web server through secure authentication and authorization schemes.

Figure 4.2 shows the interaction model of a generic web page structure. In this Figure, a web page is the main entity that contains different types of interaction elements I. The web page can be dynamic or static. A dynamic web page generates different web contents depending on the information provided by the user.

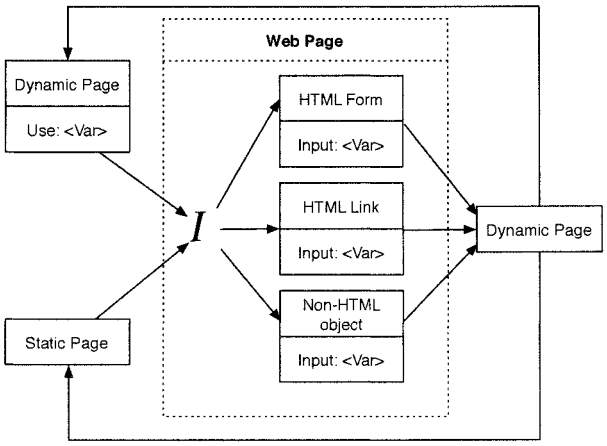


Figure 4.2: Interaction model of a web page

The basic interaction element is a form, which is specified in the HTML `<Form>` tag. This tag includes two attributes: the `action` attribute specifies the SP of a form element, while the transfer mode T is specified within the `method` attribute of the `<Form>` tag. Another type of interaction element is a link, which is represented using the `<A>` tag. SP is specified in the `HREF` attribute within the `<A>` tag, while the transfer mode T of a link is `GET` because a link only has one

value. Therefore, `GET` is specified by default (4; 7). Definition (4.1) is used to simplify the analysis of a web page through extracting the parameter values (the user input data (IT, SP, D, T, U)) of the interaction elements in the target web page by the integrity verifier component of the proposed framework (see Figure 4.1) before processing a request.

(4.1) $I \cong (IT, SP, D, T, U), D \cong \{(p_j, v_j) \mid j = 0, \ldots, n\}$

We donate a zero entry with (p_0, v_0) where $p_0 = v_0 = 0$. A generation of web content units involve interaction elements of the three basic parameters: IT, SP, and T. Therefore, we are interested to extract the values of IT, SP, and T because we focus on the integrity of the data and not on whether the user U is authorised or not (see `assumption` 2 in Section 4.2.1), and sent data D is validated or not (see `assumption` 2 in Section 4.2.1).

The injection attacks such as SQL and scripting attacks are not example of data tampering attacks. Injection attacks are a specific type of input validation problem so that data validation can be used to enhance resistance to injection attacks (84; 106). Therefore, we are not focused on user input D is validated or not.

Therefore, Definition (4.1) is used to extract the parameter values of the interaction elements for every requested element (see Function *extractRequestedResource()* in Figure 4.10) before processing a request. Figure 4.3 illustrates an example of the HTML `<Form>` tag that contains a number of data items. It is assumed that user (UserX) types "schools in UK" in the input box of the search engine.

In accordance with Definition (4.1), we obtain from Figure 4.3 the interaction element $I \cong$ (Form, search.asp, D, POST, UserX) where for example $D \cong$ {(search_value, "schools in UK")}. The integrity verifier component extracts only the following values to take the next action: (Form, search.asp,POST).

```
<Form method="POST" action="//search.asp">
<P>Enter Data </P>
<Input id="search_value" type="text"/>
<Input type="submit"/> </Form>
```

Figure 4.3: HTML form input element

4.2.4 New Hashing Strategy

We have developed a new hashing strategy to describe hashing calculations for the referenced objects that are shared among the web pages of the target web site or the web application. This strategy applies for all web referenced objects that have been developed before use over the secure HTTP request-response model. This new hashing strategy to improve the hashing performance and minimise overhead times of the proposed WCVR system.

We have set a number of assumptions and specifications to develop our new hashing strategy. We have started with three assumptions (118):

1. **(A1)** a web site is a collection of web pages that generate units of web content.
2. **(A2)** a web page is a tree of referenced objects and each web page, whether static or dynamically generated, can have zero or more referenced objects in addition to the code of the web page itself.
3. **(A3)** a referenced object is any object that has a link within the web page such as image, audio, video and other objects.

Moreover, this strategy has four specifications (118):

1. **(S1)** a web site includes a finite set of referenced objects.
2. **(S2)** external Cascading Style Sheet (CSS) objects and external codes are treated as referenced objects.

3. **(S3)** links to web pages and referenced objects for different host servers are not specified in the range of referenced objects.

4. **(S4)** it is possible to have one or more referenced object shared among the web pages of the target web site or the web application.

Let WS denote a target web site, $WPi, i = 1, 2, \ldots, n$ be a web page, O be a referenced object and C denote the source code of a $WP, \forall C \in WP_i$, where n is the number of web pages, and m is the number of referenced objects in WS.

From assumptions A1, A2 and specification S1 we obtain:

(4.2)
$$WS \cong \{WP_i \mid i = 1, \ldots, n\}$$

From (A3) we obtain

$$O \cong \{O_j \mid j = 1, \ldots, m\}$$

From (A2, A2, S1–S4) we obtain:

(4.3)
$$WP_{ji} \cong \{(O_j, C_i) \mid j = 1, \ldots, m, i = 1, \ldots, n\}$$

Substituting (4.3) into (4.2), where $WP_{ji} \cong \{(O_j, C_i)\}$ denotes a web content related to O_j and C_i

(4.4)
$$\begin{aligned} WS &\cong \{WP_{ji} \mid j = 1, \ldots, m, i = 1, \ldots, n\} \\ &\cong \{(O_j, C_i) \mid j = 1, \ldots, m, i = 1, \ldots, n\} \end{aligned}$$

To analyse this strategy, an example is presented in Figure 4.4. WS which is a web site that contains three web page: $WS \cong \{WP_1, WP_2, WP_3\}$ and seven referenced objects $O \cong \{O_j \mid j = 1, \ldots, 7\}$.

From (4.3) we get:

$$WP_1 \cong \{(O_j, C_1) \mid j = 1, 2, 3\}$$
$$WP_2 \cong \{(O_j, C_2) \mid j = 1, 4, 5\}$$
$$WP_3 \cong \{(O_j, C_3) \mid j = 1, 6, 7\}$$

And from (4.4) we get:

$$WS \cong \{WP_{ji} \mid j = 1, \ldots 7, i = 1, \ldots, 3\}$$

So, from **(Specification 4)** we get:

$$Z = WP_1 \cap WP_2 \cap WP_3 = \{O_1\}$$

Where Z is the interaction of referenced objects.

As seen in Figure 4.4, the hash value of O_1 is the same in all shared web pages at the registry level (hashing measurement

$$\text{Hash}(O_1 \in WP_1) = \text{Hash}(O_1 \in WP_2) = \text{Hash}(O_1 \in WP_3)$$

Merkle (119) proposed a tree scheme of digital signatures. This scheme provides a different message to be signed for each node in a tree. If two or more trees reference the same object in a node then the signature of that node will be different in each tree. Therefore, our strategy is different from Merkle's in that shared referenced objects have the same signature even though they belong to different web pages. This could achieve fair balance between performance (minimising the number of calculated signatures of each referenced object in the repositories of web servers) and security. Theoretically, our new hashing strategy helps to increase the hashing performance compared with the Merkle scheme which involves the computation of the hash value for every web content attributed to a web page even though it is the same web content but it is attributed to a different web page. We have conducted a set of experimental studies to test our system which are

discussed in Chapter 6. We have found out that the new hashing strategy can utilise the performance of WCVR system.

The tables of DBMS offline-transaction table only includes one hash value for O_1. In contrast, the existing approaches calculate unique hash value for every referenced object-related to a web page in the target web site or web application. This means the hash value of O_1 which belongs to WP_1 is not equal to the hash value of O_1 which belongs to WP_2. Similarly, this applies for each shared referenced object. Therefore, the proposed hashing strategy increases the hashing performance.

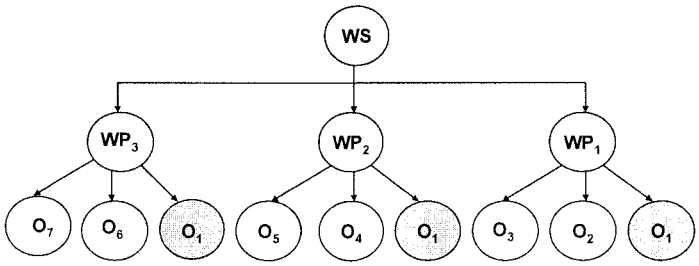


Figure 4.4: Example of a web site with 3 web pages and 7 referenced objects

Those approaches, because of their redundancy (recalculating the hash value of each referenced object even though it is shared) are more expensive and less effective in terms of the signing (hashing calculations) process than our proposed approach (93). Therefore, unlike traditional code-signing techniques, our approach allows integration a web page with other pages.

4.2.5 Functional Overview

We propose the functionality steps of the suggested solution. Before the web contents is published, the following steps are performed:

1. DBMS creates two tables to store additional details about the server-side static and dynamic web contents as a secure history. Table one is an offline-

transaction table for mapping the hash values of static web contents to their specific repositories of web servers and an online-transaction table for mapping the hash value of dynamic web content to its name of dynamic web page. Note that we have assumed that the DBMS transaction tables remain a secure.

2. A publisher or web administrator retains the original backup of all web content that have been developed for use in a secure manner for recovery purposes. Note that this type of backup is conducted offline only for all static web content, whereas we cannot make a backup for dynamic web content because we cannot predict the generated dynamic web content that relies on the user interaction. Therefore, online backup is managed by the response hashing calculator component.

3. The web register component manages the hashing calculations for all static web contents that have been developed in the repositories of target web servers for use in the secure environment. In addition, the web register sends the list of hashing values to the DBMS offline-transaction table for storing.

When a client request arrives, the following steps are performed:

1. The integrity verifier (manager) component intercepts the HTTP request (such as web page, audio, video, images, and others), checks it, analyses it, extracts the hash value of the original copy of the static web content from DBMS offline-transaction table, re-calculates the hash value of the web content, and compares the two hash values for integrity verification process. If they match, then the integrity of the requested web content is valid; otherwise, the the requested web content has been tampered with. The integrity verifier (manager) forwards the request to a web server if the enforced web policy is satisfied. If it is not satisfied, the integrity verifier sends the request to the recovery component to identify the tampering problem and reports this attack to web administrator.

2. Once a web server application has processed the request, the response hashing calculator component calculates the hash value of output response and makes a backup for the output response. The hash value of the response (dynamic web content) is appended to DBMS online-transaction table.

3. The response is intercepted by the integrity verifier component. The manager analyses the response, extracts its original hash value (this value is appended to the secured online-transaction table), re-calculates it and compares the two hash values for integrity verification process. If they match, then the integrity of the response is valid; otherwise, the response has been tampered with. Therefore, if it is not valid, the manager sends the response to the recovery component to identify the tampering problem and reports this to the web administrator.

4. The integrity verifier component forwards the correct response to the target client.

4.2.6 Security Framework Components

4.2.6.1 Web Register Component

Web register component manages the hashing measurements (calculations) for all static web contents that have been developed in the repositories of target web servers for use in the secure web environment. In addition, the web register sends the list of hashing values (checksums) to DBMS offline-transaction table for storing and maintaining. Note that modifications to the DBMS offline-transaction table are not permissible as that would enable an adversary to hide integrity-relevant actions.

If any static web content undergoes any add-ons (new modifications that are added to the original web content (93)), it is released. The updated web content must be regenerated by producing a new hash value. In addition, web register sends the list of hashing values to DBMS offline-transaction table for storing. Therefore, the web register component monitors all static web contents for any

authorised change that occurs. This requires a new measurement for every web content that is legally updated. The hashing calculation process in a web register component can take place where one of three conditions is satisfied:

1. when a new static web content is developed for use.
2. when a used element is legally changed.
3. when the digital certificate of a used element has expired.

On each modification, the DBMS obtains a checksum and computes a cryptographically strong one-way hash SHA-1 function of the (new) data of a web resource. However, some web developers may not take into account the digital certificate expiration of (used) published elements because they rely on web browsers and operating system settings. Therefore, an adversary can replace an original element with an expired original element to evade the web content of a web server or client machine (7; 12). To counteract this risk, we compute the checksum using SHA-1 hashing function in concatenation with private key, transaction id, issue date, expiration time and IP address, as shown in Figure 4.6.

To utilise the performance of a web register component, this component uses a new hashing strategy to describe how to do hashing calculations for the referenced objects that shared among the web pages of the target web site or the web application. This strategy applies for all web referenced objects that have been developed before use over the secure HTTP request-response model. For example, it is important to register a page code when there is any change, either in the viewed content, the structure of the web page or both. This is because this change causes a change in the presentation of the web page.

As regards the Notations 4.2-4.4, the process of this registration technique applies to the referenced objects and code of requested pages. Therefore, we have developed an algorithm (see Figure 4.5) that describes the functions and steps required to register a web content. The proposed algorithm can be applied

to all referenced objects and any page code regardless of the type of client or server-scripting language, and size of page code.

The algorithm in Figure 4.5 consists of steps and functions to perform hashing calculations and storage in the DBMS offline-transaction table. As shown in lines 1-7, we present the inputs of this algorithm:

1. element set E_i: each element is a web content and the element set contains three description items including identification number (ID), hash value of element ($elementHashValue$) and element path on the repositories of a web server ($elementName$).

2. secret key SK: where secret key is used to sign the registered web content element. The secret key can be distributed from secure certificate authority or it results from some mathematical equation that is conducted on the web content element. In this thesis, the secret key relies on the web content element and on other factors such as the time factor, which specifies the expiry of a secret key to identify tampering attacks.

1: **algorithm** *registeredElement*($E_i \in WS$, $SK \in CharacterSet$) **return** $elementRecordset$
2: E : Elementset /* $E = \{E_i \mid 0 \leq i \leq n\}, E_i = (ID, elementHashValue, elementName)$*/
3: $ID : LongIntegerSet$ /* Identification number*/
4: $elementContentBuffer : CharacterSet$ /* element content */
5: $elementHashValue : CharacterSet$ /* Element Hash Value*/
6: $elementName : CharacterSet$ /*Path of Element on a web server*/
7: $E \leftarrow \emptyset$
8: **foreach** $E_i \in WS$ **do**
9: $elementContentBuffer \leftarrow extractElementContent(E_i)$
10: /* *extractElementContent(elementName)* extracts the content of E_i*/
11: **if** (not *Register*($elementName$) or *Changed*($elementName$)) **then**
12: $elementHashValue \leftarrow Sign(elementContentBuffer, elementName)$
13: $Path \leftarrow extractPath(E_i)$
14: $backupCopy(elementContentBuffer, elementName, Path)$
15: $E_i \leftarrow E_i \cup (ID, elementHashValue, elementName)$
16: **end if**
17: **end for**
18: $DB \in DatabaseSet$
19: /*To store the code details in DBMS offline-transaction table*/
20: $DB \leftarrow connectDatabase()$
21: $elementRecordset \leftarrow \emptyset$
22: **foreach** $E_i \in WS$ **do**
23: $elementRecordset \leftarrow elementRecordset \cup Put(E_i)$
24: **end for**
25: Output($elementRecordset$)
26: **end algorithm**

Figure 4.5: Algorithm of registration static phase for web contents

We have also developed two basic functions that are invoked for the registration algorithm (see Lines 8-16):

1. $extractElementContent(elementName)$ function to extract streaming content element in web site WS set. Figure 4.7 includes an algorithm which describes how to extract content element.

2. $Sign(elementContentBuffer, elementName)$ function to calculate the hash value and extract digitally unique fingerprint for this element, as illustrated in Figure 4.8. In this thesis, a Message Authentication Code (MAC) has been used in our secure web environment to ensure the data integrity. MAC is used to compute the checksum of a web content in the repositories of web server. We compute a MAC using one-way hash functions (SHA-1) and a changeable secret key SK to create a special fingerprint for web content. The function in Figure 4.6 shows how to compute the MAC value using SHA-1 hash function.

```
MAC= SHA-1 ("secret key" +SHA-1 ("Transaction ID" + "issue date" +
           "expiration time" + "IP address" + "secret key")).
(Combined all transaction fields to private key and then hashed).
```

Figure 4.6: Production of MAC value

In line 14, the backup process has been established by *backupCopy*. This function makes a backup of the data files to a directory, disk or computer across the network. Web register component then monitors the source files and keeps the backup updated with new or changed files. It runs in the background with no user interaction.

Finally, as shown in lines 16- 24 in Figure 4.5, the results are stored in DBMS offline-transaction table using $Put(E_i)$ function. Table 4.1 shows a sample of expected results for the registered element in *elementRecocdset*. The schema of this *elementeRecordset* contains four elements: $ID*$ (ID is the primary key of this schema), $elementHashValue$, and $elementName$.

1: Input($elementName \in CharacterSet$)
2: **function** *extractElementContent*($Input$) **return** $contentBuffer$
3: $contentBuffer \in CharacterSet$
4: $binaryBuffer \in CharacterSet$
5: $connectFileInputStream$
6: **foreach** $binaryBuffer \in E_i$ **do**
7: $contentBuffer \leftarrow contentBuffer \cup readBinary(binaryBuffer)$
8: /*readBinary(binaryBuffer): is a function to read the content of element in binary format */
9: **end for**
10: Output($contentBuffer$)
11: **end function**

Figure 4.7: Extracting of content element for hashing calculation

1: Input($elementContentBuffer \in CharacterSet$,$key \in CharacterSet$)
2: **function** *Sign*($Input$) **return** $hashValue$
3: $hashValue \in CharacterSet$
4: $hashValue \leftarrow MAC(SHA-1, Key, elementContentBuffer)$
5: // MAC is Message Authentication Code function which use hash function (SHA-1)
6: // To produce an unique hash value
7: Output($hashValue$)
8: **end function**

Figure 4.8: Producing hash value using SHA-1 hash function and private key (MAC technology)

Table 4.1: Expected results in the DBMS offline-transaction table.

Expected Registry Table		
ID	elementHashValue	elementName
331	0FD3CF858FEA3F594F4283DFCBCC22DBB6E9526E	F:\ShoppingCart\defaultroot\apply_hash.js
332	25619C866BEEE6A195A2B41A71236DD915BD5D4A	F:\ShoppingCart\defaultroot\arrow_rm.cur
333	DE78CF892345444F06ED056A2FFA20665342937E	F:\ShoppingCart\defaultroot\Cart.jsp
334	C5AB00CBD0515C3199D0250B3A4202AEB22E9C2D	F:\ShoppingCart\defaultroot\cookie.js
335	BA5EBC53317BD4A6E684F63F8CE07E63EC8A1659	F:\ShoppingCart\defaultroot\CookieformBean.class
336	1479BBA3E9A97F2C0544F8DBA34B5AD6446B20DD	F:\ShoppingCart\defaultroot\Copy of stuff.js
337	6720DD40EC32AA46BB7A24F022AC9A77AF54024D	F:\ShoppingCart\defaultroot\Copy of ThankYou.jsp
338	88202B9818C0ABE6C98A5437BEE467008EFF77E0	F:\ShoppingCart\defaultroot\fpdbform.inc
339	8751D4DAF5939E807FFF93769CB827CE536953AD	F:\ShoppingCart\defaultroot\frm_linevrlt.png
340	176984926A7808D2A6C38F3F1DB8A6EE28181CB5	F:\ShoppingCart\defaultroot\images\books.gif
341	EC0F58F638BCF42DFE7066B94B8CC9109C9A5997	F:\ShoppingCart\defaultroot\images\dollar.gif
342	807FC2A1B0F280E60A1AE911812376AF2A1CF954	F:\ShoppingCart\defaultroot\images\greenpaper.gif
343	653836D6DEA43191EE52F7D4882554C9D493A8F4	F:\ShoppingCart\defaultroot\images\larrow.gif
344	BBD863C902184F1D04D6317DA451C1AA57401A2E	F:\ShoppingCart\defaultroot\images\rarrow.gif
345	D510D5D5E6812962625CD363DFD4FC858477A775	F:\ShoppingCart\defaultroot\images\Thumbs.db
346	A389151D2DF6DFE1A764BB4C4ECBB887296DAB5A	F:\ShoppingCart\defaultroot\Purchase.jsp
347	BB8E71E3635BFBF0333604F72F316E44A76A948C	F:\ShoppingCart\defaultroot\rmic.exe

4.2.6.2 Integrity Verifier Component

The integrity verification aims to detect the alterations in the verification data, which should be significantly much smaller than the multimedia data, can be stored in secure DBMS tables.

The integrity verifier component (manager) is positioned between the client machines and the target web server. This component manages the HTTP requests and responses via a state protocol that enforces a number of web policies that apply to the elements of the web-based system. The web policy specifies the next action or decision to be taken whether the request or response is valid or not and when the policy should be enforced. This component executes a state protocol to dynamically generate a collection of enforced web policies.

The integrity verifier component launches an online state protocol to enforce a set of web polices, as shown in Figure 4.9. The web policy specifies the next action or decision to be taken whether the request or response is valid or not and when the policy should be enforced. In this thesis, we have four web polices as follows:

- request availability policy (policy 1): is used when the requested resource is available at the repositories of target web server(s).
- integrity failure policy (policy 2): is used when a tampering attack is detected during integrity verification process.
- integrity passing policy (policy 3): is used when the web content has not been tampered with.
- recovery policy (policy 4): is used when the tampered web content has been recovered.

Note that the outputs of web servers, as well as requests are examined in our integrity verification process, as shown in Figure 4.9.

As an integrity check, this state protocol is similar to adaptive intrusion-tolerant server systems (110). Figure 4.9 shows the finite automata of the state

protocol that contains a number of steps to generate the enforced web policies. The manager comprises the following stages when enforcing the correct web policy.

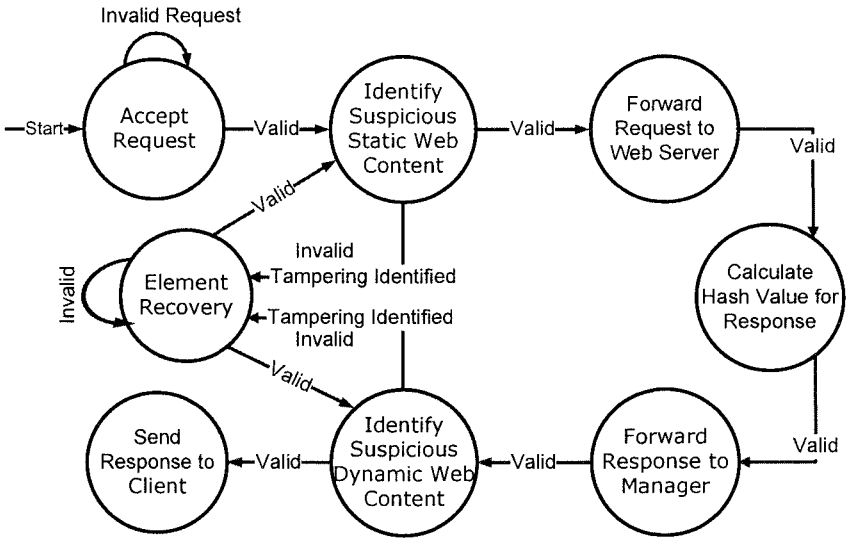


Figure 4.9: The finite automata of state protocol

We have proposed two levels of integrity verification:

1. The integrity verification at the request level for verifying a static web content.

2. The integrity verification at the request level for verifying a dynamic web content.

The integrity verification at the request level has two phases where each phase has a number of tasks.

1. The integrity verifier (manager) component intercepts the HTTP request. Function *extractRequestedResource()* in Figure 4.10 illustrates how to extract the requested resource *SP*, based on the derived Definition (4.1). For

example, in a web form, we extract the path of a requested resource from the `Action` attribute.

2. At this point, the state protocol enforces the request availability policy which is used when the requested resource is available at the repositories resources. Figure 4.12 illustrates how to check whether or not the requested resource *SP* is available at the repositories of web servers. If this policy is satisfied, the integrity verifier passes to integrity verification process. Otherwise, the integrity verifier sends back a message to target user that this requested resource does not exist.

3. The next web policy is the integrity failure policy (policy 2) which is used when a tampering attack is detected during the integrity verification process. The verification process contains the following steps:

   (a) re-calculating the hash value of a requested resource *SP*. The following algorithm in Figure 4.7 shows how to extract the content of *SP* and then sign computationally the content of *SP* to retrieve an unique checksum, as shown in *Sign* function in Figure 4.8.

   (b) extracting the hash value of the original copy of the static web content from DBMS offline-transaction table.

   The function *extractRegisteredHashValue* in Figure 4.11 illustrates how to extract a checksum of registered content element from DBMS offline-transaction table. Line 12 outputs $originalHashValue$, which specifies a a checksum of registered element.

   (c) comparing the two hash values for integrity verification process. If they match (policy 3 is satisfied) then the integrity of the requested web content is valid and the integrity verifier then forwards the request to a web server. Otherwise, the requested web content has been tampered with (policy 2 is satisfied) and the integrity verifier then sends the request to the recovery component to identify the tampering problem and reports this attack to a web administrator. The function shown in Figure 4.13 illustrates a comparison of two elements to produce the correct policy.

1: Input($I_j \in I$)/*I is the interaction element*/
2: **function** *extractRequestedResource*($Input$) **return** *requestedPath*
3: $requestedPath \in CharacterSet$
4: $I_j \leftarrow (IT, SP, D, T, U)$
5: $requestedPath \leftarrow I_j(SP)$
6: Output($requtestedPath$)
7: **end function**

Figure 4.10: Extraction of SP path

1: Input($requestedElement \in CharacterSet$)
2: **function** *extractRegisteredHashValue*($Input$) **return** $originalHashValue$
3: $originalHashValue \in CharacterSet$
4: $originalHashValue \leftarrow \varnothing$
5: $DB \leftarrow connectDatabase$
6: **foreach** $record \in Recordset$ **do**
7: **if** ($requestedElement \in recordset$) **then**
8: $originalHashValue \leftarrow record(elementHashValue)$
9: Exit loop
10: **end if**
11: **end for**
12: Output($originalHashValue$)
13: **end function**

Figure 4.11: Extraction of original hash value

```
1: Input(requestedPath ∈ CharacterSet)
2: function extractrequestAvailabilityPolicy(Input) return
   requestAvailabilityPolicy
3:   requestAvailabilityPolicy ∈ CharacterSet
4:   foreach WP_i ∈ WS do
5:     if (requestedPath ∈ WP_i) then
6:       requestAvailabilityPolicy ← "Valid"
7:       Exit Loop
8:     else
9:       requestAvailabilityPolicy ← "Invalid"
10:    end if
11:  end for
12:  Output(requestAvailabilityPolicy)
13: end function
```

Figure 4.12: Enforcing request availability policy

```
1: Input(requestedHV ∈ CharacterSet,originalHV ∈ CharacterSet)
2: function matchHV(Input) return integrityRequest
3:   integrityRequest ∈ CharacterSet
4:   if (requestedHV ≐ originalHV) then
5:     integrityRequest ← integrityPassingPolicy
6:   else
7:     integrityRequest ← integrityFailurePolicy
8:   end if
9:   Output(integrityRequest)
10: end function
```

Figure 4.13: This function compares between the original checksum and the re-calculated checksum for integrity verification of static web content

To minimize the performance overhead for hashing calculations at the request level, we eliminate future measurement computations as long as the web content has not been altered. The process of our integrity verification (including the hashing calculations) is based on the principle of HTTP request-response model which declares the statelessness of HTTP (each client request results in a new connection between a web browser and a web server). In other words, our integrity verification takes into account how the HTTP requests are processed where each explicit or implicit request is served separately. To analyse this, Figure 4.14 takes for example which is presented in Figure 4.4. This Figure illustrates how the hashing calculation in our system works.

We use the RFC of HTTP protocol (44). Therefore, there is no waiting to calculate the hash values for referenced objects of requested page. Each implicit or explicit request has been verified separately and then it sends to a web browser without waiting other objects of the target web page. As a result the performance overhead of integrity verification process is minimised. In contrast, existing systems such as DSSA identifies waiting to calculate the final hash value for the requested web page and all their referenced objects for integrity verification purposes.

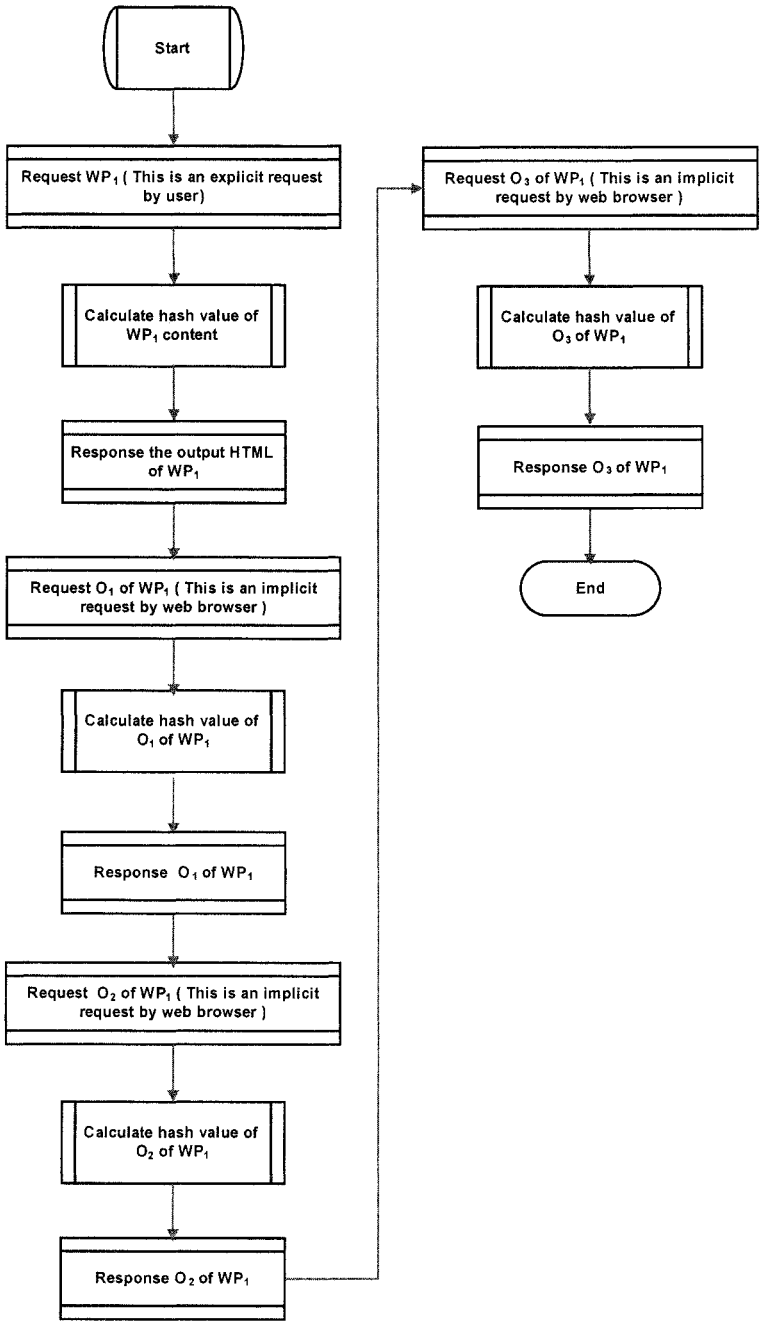


Figure 4.14: The process of the hashing calculation in our WCVR system

4.2.6.3 Response Hashing Calculator

A response hashing calculator is an extension to the web server application. When the web server application processes a request, response hashing calculator is invoked. Once the response hashing calculator is invoked, it makes a backup for each output response if a web resource is a server-side dynamic page. In addition this component calculates the hash value of an output response before sending it back to the manager. The hash value of the response (dynamic web content) is appended to DBMS online-transaction table.

If policy 4 is satisfied, a response is sent back to the client. The following algorithm in figure 4.15 shows how to extract the content element of a responded web page. This algorithm takes place after processing a request on a web server.

In line 15, the backup process has been established by *onlineBackupCopy*. This function offers an online backup of HTTP generated responses to a directory, disk or computer across the network. The backup function helps to protect data by restoring files on the designated directories of the web server. In this way, if the server-side web content crashes or is infected by a malicious application that results in loss of data, access to data on the backup disks is still possible.

Table 4.2 shows a sample of expected results for the registered response in a *responseRecordset*. The schema of this *responseRecordset* contains four elements: $ID*$ (ID is the primary key of this schema), $responseName$, and $responseHashValue$.

Table 4.2: Expected result in the response table

Expected Response Registry Table		
ID	responseName	responseHashValue
466	/StartHere.jsp	A337E22716EEF3D503192A07743E4B484189CEFE
467	/Store.jsp	689E8EB9373FE84313634F5E7932146D5EC94B4E
468	/Cart.jsp?num=2&count=6	1920D455AFE02BE62487757AB6054E339A364775
469	/Purchase.jsp	7A6F34DD52A4B6C21099D8A114710446F344D7CC
470	/Cart. jsp?num=4&count=6	1920D455AFE02BE62487757AB6054E339A435677
471	/ThankYou.jsp	4487CA7216AF5021E195C7BB48943DB42895B527

1: **algorithm** *registeredResponse*(*response* $\in$ *CharacterSet*, *SK* $\in$ *CharacterSet*) **return** *responseRecordset*
2: R : Responseset /* $R = \{R_i \mid 0 \leq i \leq n\}, R_i = (ID, responseHashValue, responseName)$*/
3: ID : *LongIntegerSet* /* Identification number*/
4: $responseContentBuffer$: *CharacterSet* /* response content */
5: $responseHashValue$: *CharacterSet* /* Response Hash Value*/
6: $responseName$: *CharacterSet* /* Response Name*/
7: $R \leftarrow \emptyset$
8: **foreach** $R_i \in WS$ **do**
9: $responseContentBuffer \leftarrow extractElementContent(R_i)$
10: Check type of $responseContentBuffer$
11: /* *extractElementContent(responseName)* extracts the content of R_i*/
12: $responseHashValue \leftarrow Sign(responseContentBuffer, responseName)$
13: $responseName \leftarrow extractResponseName(R_i)$
14: $R_i \leftarrow R_i \cup (ID, responseHashValue, responseName)$
15: $onlineBackupCopy(responseContentBuffer, responseName)$
16: **end for**
17: $DB \in DatabaseSet$
18: /*To store the response details in DBMS online-transaction table*/
19: $DB \leftarrow connectDatabase()$
20: $responseRecordset \leftarrow \emptyset$
21: **foreach** $R_i \in WS$ **do**
22: $responseRecordset \leftarrow responseRecordset \cup Put(R_i)$
23: **end for**
24: Output($responseRecordset$)
25: **end algorithm**

Figure 4.15: Algorithm of response registration

4.2.6.4 Integrity Verifier Component at the Response Level

The response is intercepted by the integrity verifier component. The manager analyses the response, extracts its original hash value $originalHV$ (this value is appended to the secured online-transaction table), re-calculates its $responseHV$, and compares the two hash values in the integrity verification process (see Figure 4.16). If they match, then the integrity of the response is satisfied; otherwise, the response has been tampered with. Therefore, if policy 2 is satisfied, the manager sends the response to the recovery component to identify the tampering problem and reports this to the web administrator.

The integrity verifier component forwards the correct response to the target client if the policy 3 is satisfied.

1: Input($responseHV \in CharacterSet$,$originalHV \in CharacterSet$)
2: **function** $matchResponseHV(Input)$ **return** $integrityResponse$
3: $integrityResponse \in CharacterSet$
4: **if** ($responseHV \doteq originalHV$) **then**
5: $integrityResponse \leftarrow integrityPassingPolicy$
6: **else**
7: $integrityResponse \leftarrow integrityFailurePolicy$"
8: **end if**
9: Output($integrityResponse$)
10: **end function**

Figure 4.16: This function compares between the original checksum and the re-calculated checksum for integrity verification of dynamic web content

4.2.6.5 Recovery Component

The Recovery component recovers the tampered web content if the action of the enforced web policy 2 from the integrity verifier is satisfied. If the integrity verification check fails, it is sent to the recovery component, which tries to extract the original web content that is known to be safe. Once it is determined that the web content has since been modified in an unauthorised manner, the system will try to recover the original web content, put it in a new assembly and discard the tampered web content. When the new assembly is generated, the recovered assembly is sent to its direction, which accesses the functionality it needs and execution continues as normal.

In this thesis, it is important to back up of the data. The backing up of data is the ability to recover that same data and recover it in a timely state to keep a service up and running. It should be noted that each backup copy (at the request and response levels) has unique name for searching purposes. In the case of static content, the web register component makes a backup copy for every static content that has been registered (i.e. register a backup copy in the offline-transaction table by generating a new assembly of hash value). This backup copy stores in the secure server backup for recovery purposes.

In the case of dynamic content, the response hashing calculator makes a backup copy for every generated web content on the fly before it is served to a client. This backup copy stores in the secure server backup for recovery purposes. In addition, the generated web content is hashed and the hash value stores in the DBMS online-transaction database (i.e. registering a backup copy in the online-transaction table by generating a new assembly of hash value).

The steps of recovery process for a web content are as follows:

1. If the detection of altered web content happens at the request level;
    (a) get backupCopy (Extract request details, get the request name, search using the unique name in the secure server backup) as as shown in line 14 - Figure 4.5;

(b) verify backupCopy from tampering (by the integrity verification process as shown in Section 4.2.6.2)

(c) forward backupCopy to a web server; and

(d) store backupCopy in the web server root.

2. If the detection of altered web content happens at the response level;

(a) get onlineBackupCopy (Extract response details, get the response name, search using the unique name in the secure server backup) as shown in line 15 - Figure 4.15;

(b) verify onlineBackupCopy from tampering (by the integrity verification process as shown in Section 4.2.6.2); and

(c) forward onlineBackupCopy to a client.

3. Discard the tampered web content.

4. Generate a log file (This file contains details such as the request information; the response information; the original hash value; the re-calculated hash value; the state of database; the state of verification process; and the state of the recovery process).

4.3 Threat Model

We have supposed that all network communication is performed through secure protocols such as SSL, as shown in Section 3.4.1. An adversary can penetrate the web system in many forms (120; 121). An insider adversary who gains physical access to a web server is able to to destroy any type of static content in the root of a web server. It is not only physical access to a server that can corrupt a web system. A malicious web content manipulation software can penetrate a server machine and once located on the server such malicious software can monitor, intercept, and tamper online transactions in a trusted organisation. The result typically allows the adversary full root access to server data and web server application. Once such access has been established, *the integrity of any*

data or software on a server is in question: the source code of web applications and web sites, the data in the database and the log files could all have been tampered with. It should be noted that the information of hash values and other details are stored in secure DBMS tables and to remain secure and unaltered (see Section 4.2.6.1).

Even though adversaries gain access to static or dynamic web content, and they change the data of particular web content, they cannot manipulate the hash value (checksum) which is stored in secure DBMS tables so the web content remains valid. This is because the WCVR system uses a cryptography hash function where the hash function is one-way (only forward) (86; 87; 122). Note this can hold even when adversary has access to the hash function itself and the hash value of original web content which is assumed to be stored in secure DBMS tables. The adversary can instead compute a new hash value for the altered web content but that hash value will not match the one that was requested. Therefore, the proposed recovery component in WCVR system sends the original requested web content to a client.

As an integrity check, the proposed state protocol is similar to adaptive intrusion-tolerant server system (110). For each web resource whose integrity is to be checked, a checksum (hash value) is computed from a private key. To resist possible guesses by an adversary, the checksum is computed by applying a one-way SHA-1 hash function to the concatenation of the private key and the content to be checked. The resulting checksum is then compared with a pre-computed one. This is sufficient to check if a web content has been tampered with.

However, we must also consider the possibility that an adversary with complete control of a web server modifies the web server application to return a correct response for a web content incorrectly modified. In particular, if the private key is always the same, it is easy for the adversary to pre-compute all responses, store the results in a hidden part of the memory and then modify the web content. An adversary would then be able to return correct responses for incorrect web content. Mihçak et al. (123) describe that using the same secret key for a number

of different items such as audio items may compromise security since each item may leak partial information about the hash value.

Therefore, we could guarantee the freshness of the request and response computations by using a changeable private key. The integrity verifier component could check that each response corresponds to the specified private key by keeping a local copy of all sensitive files and running the same computation as the server but this imposes an extra administrative and computational load on the manager.

In this thesis, we compute a SHA-1 hash with changeable private key over the complete contents of the web resources. The resulting 160 bit hash value unambiguously identifies the resource's contents. Different resource types, versions and extensions can be distinguished by their unique checksums.

4.4 Conceptual Comparison

In order to address the limitations and weakness of the existing approaches, the proposed WCVR system needs to consider the following:

1. To ensure the integrity of a static web content against unauthorised tampering. The WCVR system should be able to detect all kinds of tampering problems such as defacement of web page and reference objects and visual spoofing attacks at the request level.

2. To ensure the integrity of a dynamic web content against unauthorised tampering. The WCVR system should be able to detect all kinds of tampering problems at the response level.

3. To provide a recovery component that should be able to recover any altered static and dynamic web content. Therefore, the survivability of web content can be achieved.

In addition, the WCVR system shows a number of advantages over other approaches:

1. Unlike adaptive intrusion-tolerant server architecture, the WCVR architecture is not complex. The WCVR architecture does not involve redundant servers running on diverse operating systems and various operating systems.

2. Unlike the performance of Dynamic Security Surveillance Agent (DSSA) and adaptive intrusion-tolerant server system, the WCVR increases the end-of-end performance by utilising a model for hashing strategy that could minimise the latency for online verification.

3. Unlike other existing approaches, the WCVR system does not require modifications to existing web application architectures. It does not also require any additional changes on the client and server and it is compatible with all major web browsers.

4. Unlike all existing approaches, the WCVR system would be able to analyse the code at the request and response level. This analysis will be able to verify the integrity of dynamic web content. Therefore, the WCVR system would not be a blind system that is unable to understand the HTTP request and response across a network.

5. Unlike all existing approaches, the WVCR system would also verify the dynamic web server content as well as the static web content.

6. Unlike all existing approaches such as DSSA, the WCVR system supports a recovery component to recover only the altered web content so it does not need to recover the whole generated static and dynamic web content for a requested web page.

7. Our proposed system does not rely on a database of known tampering attacks.

8. Unlike client-side encryption approach and application-level gateway approach, the WCVR system supports DBMS tables to store and maintain for every details about web content instead of adding the hash value between the tags (such as `Meta` tag) of HTML or XML response, and as result, the malicious web content software could intercept and analyse to obtain

the original hash value from the output response so it is easy to evade the web system by tampering the original hash value of web content.

Furthermore, it should be noted:

1. The WCVR system does not validate the user information either on the client or server because we would use the form validation scheme to validate the user information. Therefore, we recommend that the validation modules to be operated on the client and server because if the client validation modules is subverted and the server validation modules are still in work.

2. As described above, ensuring the integrity of user information is not sufficient to ensure survivability of the whole web system. Therefore, WCVR system would also use the SSL protocol, form validation modules and firewalls.

There is always a tradeoff between the security and performance. We focus on the integrity check on the original element to detect malicious codes added in an unauthorised manner. The proposed WCVR system could detect the page code and every referenced object on the designated directories of web server and web content that is generated on the fly against tampering problems. The attackers are interested in targeting the referenced objects and page code on the fly. For example:

- It is possible to replace any original image by another image that contains malicious code where a victim requests the altered image and then it can destroy a web server or client machine.

- For example, it is possible to create a Java utility that can listen, intercept, monitor and capture both client request and server response this because communication channel between a client and a web server is conducted with streams and sockets (124).

- The Cascading Style Sheet (CSS) object is threatened through a visualisation spoofing attack. The strategy of this attack is to change any important information that is identified by a particular colour to another colour. The objective is to manipulate the user into making a decision that is based on incorrect information.

WCVR automatically checks for the following vulnerabilities:

- visualisation spoofing attack.
- textual spoofing attack.
- web application verify.
- tampering code manipulation (source code, path, and link).
- tampering object manipulation(audio, images, video, and other referenced objects).
- defacement of the web page.

4.5 Conclusion

The current solutions are not sufficient for ensuring the survivability of server-side static and dynamic web content. For instance, the SSL protocol is unable to ensure the data integrity at the ends of web system (such as web browser and web server) (125). In addition, the current technologies such as input validation schemes, firewalls, cryptography and access control are not capable of verifying the integrity of web content before a request or response enters the secure communication channel because a malicious code can be installed behind firewalls.

In an attempt to overcome the research problem, we have proposed a novel survivable system that could provide continued and correct services to internal and external users, even though a web data manipulation problem may have occurred. The proposed framework includes a new state protocol to enforce a

set of web policies, and a supporting software system to verify server-side static and dynamic web content before the client receives the requested page. This approach would add confidence in terms of users correctly accessing web services and displaying electronic materials on their web browsers.

We have also compared our WCVR system to prior and related work. In this thesis we have described how our proposed system can continue to function in case of an attack on a web content it is trying to use.

In the next chapter, we present the implementation of WCVR system and its proposed mechanisms. The tools used in creating the prototype are discussed and the architecture of the prototype are depicted. In addition, chapter 5 will describe the experimental design and pilot study. Thus, we design five experiments to meet the security and performance objectives.

Chapter 5

Implementation of WCVR System and Initial Testing

Chapter 4 has described the design of WCVR system for investigating survivability of server-side static and dynamic web content against tampering attacks. The WCVR system has been constructed from a web policy framework of a number of components, the client interaction elements model, the new hashing strategy and work assumptions. In addition, chapter 4 has explained the steps of the proposed state protocol to enforce the appropriate web policy at the response and request levels. Furthermore, we have compared the WCVR system with other existing systems, approaches and protocols.

In this chapter, we will describe the implementation of our proposed system. The WCVR system is implemented in Java, Servlets and Filters. The DBMS Microsoft Access 2007 Database is selected as the repository for storing and retrieving details about static and dynamic web contents. In order to demonstrate that our WCVR system is able to ensure the survivability of server-side web content against tampering, we have undertaken some experimental testing. For experimental design, we have used a local network of two HP server machines: web server and database server, two routers, two switches and four client machines. The web servers used are Apache 1.3.20 running on MS Windows Server 2003, Microsoft IIS 6.0 running on MS Windows Server 2003 and Apache Tomcat

5.01 on MS Windows Server 2003.

Section 5.1 will describe the tools used for the implementation of WCVR prototype. Section 5.2 will illustrate a high level architectural design showing the main components and modules of the prototype. In addition, it will describe components of the main modules of the prototype and explain how the components work and communicate with each other. In Section 5.3, we will define a testing strategy to evaluate the WCVR system to meet the security and performance objectives. Section 5.4 will discuss the pilot study in some detail.

We will show with an implementation and a pilot study, that the overhead for verification and recovery process are relatively low and that the WCVR system can efficiently and correctly determine if the server-side static and/or dynamic web content has been compromised. Section 5.5 will offer conclusions on the implementation of WCVR system, testing strategy and results of a pilot study.

5.1 Tools used for the Implementation

A number of tools are involved with the implementation of the WCVR prototype as follows: Java, Servlets and Filters. The reasons for choosing these tools are given below.

5.1.1 Programming Languages for Implementing the Prototype

5.1.1.1 Java

Currently, several programming languages are used, we have selected Java to implement the prototype of our WCVR system since (see Section 2.3.4):

- Java is a platform independent and several benchmarks (78) has shown that Java has acceptable performance compared with other programming languages such as C and C++ (126).

- Java was designed to work in a network-computing environment (see Section 2.3.4).
- Java supports the code signing using JDK 1.1 for securing the integrity of Java Applets (24; 27).
- Java includes Java Virtual Machine (JVM) platform to provide a trusted environment for running the Applets, which are embedded in a web page. However, JVM cannot secure a web page against a malicious Applet (80).
- Java supports multi-threaded applications[1].

5.1.1.2 Servlets

We have used Servlets in our implementation. Servlets can be written in some programming languages such as Java, and C#. In this thesis, a Servlet is a Java program that runs on a web server. We have chosen the Servlets in Java for the following reasons:

- Java supports a Servlet application that works over a web system. Servlet is used to generate dynamic web pages on demand (each time, the page is requested) (27).
- The Java Servlets includes `HTTPservlet` class that manages the multiple requests. This feature solves the sessionless HTTP drawback (28). Therefore Servlet is developed instead of CGI for better scalability and security (36; 38).
- Java Servlets is a standard extension to the Java platform for writing reliable and portable web applications (40; 127).
- Along came Java Server Pages (JSPs) introducing an easier way to generate HTML (127).

[1]Multi-threaded applications allows different parts of a application to run concurrently. Further details can be found in www.intel.com/products/glossary/body.htm.

5.1.1.3 Java Filters

The Servlet 2.3 Application Program Interface (API[2]), now a proposed final draft, has been released and it brings with it another powerful feature called Filters. The Filters are implemented by Servlet containers (127).

Java Filters permit a declarative pre- and post-processing of requests and responses handled by web resources such as Servlets, JSP, HTML files and even other Filters in a web application (127). Where declarative means the Filter can be applied in a deployment descriptor of a web application or applied in a web adminstration tool rather than programmatically in a Servlet or JSP. Figure 5.1 show how the Java Filters which positioned between the requested recourses to monitor, and intercept each request and response for various purposes.

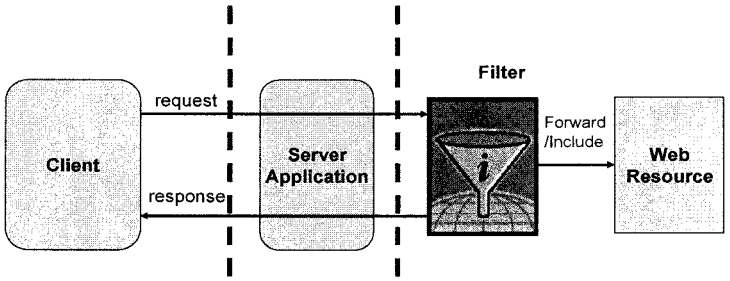


Figure 5.1: Filter Process

Java Filters have several capabilities as follows (127):

- They are able to intercept requests to one or more web resources to perform some actions,
- Modify the request header, and
- Pass a request to the next Filter in the chain or to the requested resource such as a server application or a web browser.

[2]A set of calling conventions defining how a service is invoked through a software package. Further details can be found in www.epcc.ed.ac.uk/other-information/glossary/.

- They are also able to intercept the response, perform some action, modify the response, pass it on and block it. In this thesis, we have used the Java Filters for this beneficial capability in implementing response hashing mechanism.

Other potential capabilities of Filters include (127):

- Authentication - if not logged in, Filters are directed to a login page.
- Logging - logs the URL and request parameters and/or all HTML output.
- Content screening - checks for disallowed content in request parameter's values and/or response output. The request or response could be blocked or the malicious content could be deleted.
- Browser cache blocking - prevents pages from being cached in the browser by modifying the settings of response headers or adding the appropriate HTML.
- Compression or encryption - response can be compressed or encrypted.

5.2 Architecture Design of the Prototype

The WCVR prototype consists of three mechanisms: Web register, HTTP interface, and response hashing mechanism. The high level architectural flowchart in Figure 5.2 depicts components of the main modules of the prototype and explains how the components work and communicate with each other.

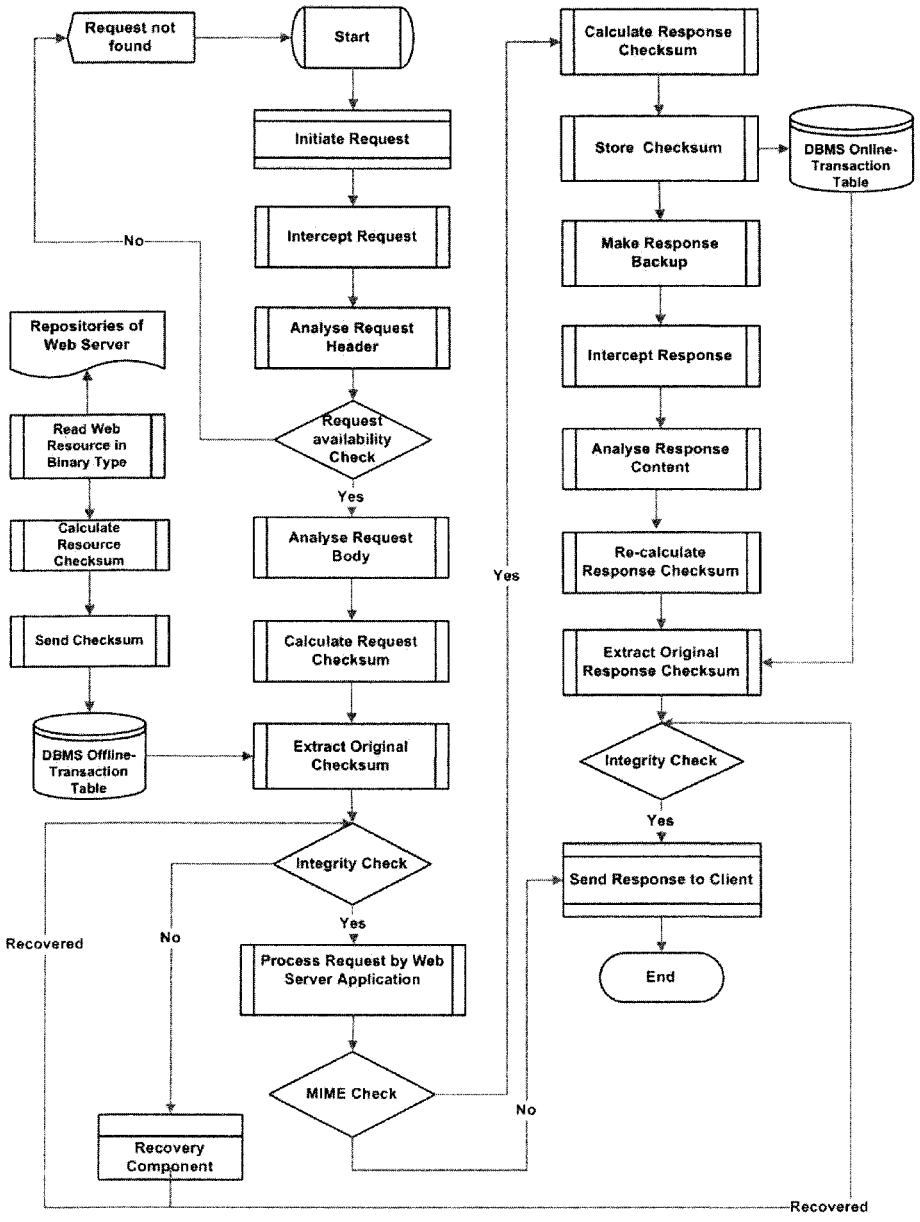


Figure 5.2: High level architectural flowchart of WCVR prototype

5.2.1 Web Register Mechanism

The functionalities of the web register mechanism are summarized into the following:

- Reading in binary format for every static web content in the secure repositories.
- Calculating the hash value for every static web content using SHA-1 function.
- Requesting Microsoft Access DBMS to store details for every static web content.
- Checking the modification status for every static web content, if modified, recalculate the hash value with taking into account the new assembly of a private key which is used in SHA-1 hashing function.

5.2.2 Response Hashing Mechanism

We have implemented the response hashing mechanism by Filters and Servlets in Java. This mechanism aims to calculate the hash value of the output response (dynamic web content) which is generated by server scripting language such as JSP, ASP, PERL, and others, and to make online backup for the output response (the produced dynamic web content) in a secure server repositories. The hash value of dynamic content is stored in the DBMS online-transaction table for integrity check before the client receives the requested page.

The main classes and interfaces that have used in the response hashing mechanism include:

1. `Filter` - the main interface to implement.
2. `FilterChain` - it passed to the Filter in its `doFilter()` method. It is used to call the next Filter (or target resource) in the chain.

3. `FilterConfig` - it passed to the Filter in its `init()` method. It is used to retrieve Filter initialization parameters and to retrieve the `ServletContext` object for the web application.

4. `ServletResponseWrapper` - generic wrapper allowing interception and modification of the response.

5. `HttpServletResponseWrapper` - HTTP specific response wrapper.

When a `Filter` is loaded, the `init()` method is invoked. We have saved a reference to the `FilterConfig` object to retrieve an instance of the `ServletContext` object. It could also have been specified as a Filter initialization parameter in the deployment descriptor of web application (a listing of web.xml in Figure 5.3). Once a `Filiter` is loaded and the `init()` method is invoked, the Filter can process requests. Each request that should be filtered calls the `doFilter()` method.

The `doFilter()` method first extracts the request object. This class only handles HTTP requests, so the next step checks whether the request coming in is an HTTP request by checking the type of the response object (`HttpFilter` interface). In each check, it must also explicitly cast the response to an `HttpServletResponse` type because the custom response wrapper in this class `OutputCaptureResponseWrapper` extends `HttpServletResponseWrapper`, which wraps an `HttpServletResponse`. An instance of `OutputCaptureResponseWrapper` is then created so that it can be passed along in place of the regular response object.

In this `FilterChain` object, there is only one Filter that calls `doFilter()` method so that the target web resource is invoked. The target resource emulates producing dynamic content. This means it just forwards a request to a static HTML file containing the target web content.

The `OutputCaptureResponseWrapper` class uses a custom `ServletOutputStream` invoked `ByteArrayServletOutputStream`. These two classes allow the capture and retrieval of the response output.

Then, we have invoked the `doFilter()` method and have captured the content produced by the resources, thus we can retrieve the content and calculate the hash

value of this content by SHA-1. The buffer is flushed to make sure that all the output as a `String` or `byte` array when it is retrieved. If the content type is "`text/html`", we will perform three actions:

1. online backup of the output response.
2. calculating the hash value of output response.
3. storing details (such as hash value) in a secure DBMS online-transaction table.

Otherwise, it just lets the content pass through.

Wrapper classes are used when the Filter must modify what the request or response returns. For example, in our `OutputCaptureResponseWrapper`, we have needed to override the methods related to the OutputStream and Writer used for outputting content. It should be noted that the `getOutputStream()` and `getWriter()` methods have returned our custom classes (which captured output), rather than the standard. The target resource can then be using our custom OutputSream and Writer.

5.2.2.1 Deployment

To invoke the main Filter class in our response hashing prototype, we have needed to deploy this through the deployment descriptor (`web.xml`) (128). Figure 5.3 shows a listing of the deployment descriptor (`web.xml`) for this Filter example.

The client's web browser requests the URL as follows:

http : //192.168.10.1/*test.html*

And the Apache Tomcat web server looks up "192.168.10.1/test.html" using configuration data provided in the web.xml file for this application, as shown in Figure 5.3. Once a `Filiter` is loaded, the Filter can process requests. This configuration data gives the complete package and class name.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
  Copyright 2004 The Apache Software Foundation

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  -->

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">

<display-name>Welcome to Tomcat</display-name>
<description>Welcome to Tomcat </description>

<!-- JSPC servlet mappings start -->

<filter>
<filter-name>responseHashingCalculator</filter-name>
 <filter-class>shoppingcart.responseHashingCalculator</filter-class>

</filter>
<filter-mapping>
<filter-name>responseHashingCalculator</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>

<!-- JSPC servlet mappings end -->

</web-app>
```

Figure 5.3: The "`web.xml`" for Response Hashing Deployment.

5.2.3 HTTP Interface Mechanism

The HTTP interface mechanism is the manager of the WCVR system based on the integrity verifier component. This component launches a state protocol to enforce the target web policy. The value of web policy determines the action(s) that should be taken by the HTTP interface mechanism. All the web policies have been implemented in this mechanism. The goals of HTTP interface mechanisms are:

- Online verification of integrity of server-side static web content.
- Online verification of integrity of server-side dynamic web content.
- Online recovery of server-side static web content if the static web content has been tampered with.
- Online recovery of server-side dynamic web content if the dynamic web content has been tampered with.

The HTTP interface prototype takes advantage of the fact that web browser requests are directed at both a specific host and a specific port. In this thesis, the Apache Tomcat and Microsoft IIS web servers listen on port 8081. The HTTP interface mechanism listens for browser requests on a default port 80 and redirects to Tomcat and to IIS. Responses coming to this mechanism are both sent to the client on port 80 and copied to a secure repository for recovery purposes.

In this thesis, we have developed a multi-threaded java application for handling concurrent connections (requests in parallel) using multiple threads that increase the power and flexibility of a web server and client programs significantly.

The HTTP interface mechanism is a multi-threaded server application that creates a new thread for every client request and another thread for every server response because this mechanism aims to intercept, analyse and monitor every request and response to investigate tampering attacks on server-side static and dynamic web content. Each client has its own TCP connection to a web server

for passing data back and forth. We have used the implementation of a version 1.0 of HTTP (44), where separate HTTP requests are sent for each web resource in a web page. Therefore, the HTTP interface mechanism establishes a TCP connection to a web server on a particular host and port number in its `httpServer` class. The `httpServer` class creates a `Socket` object with a computer name (the IP address of a server: 192.168.10.1) and port number (such as 8081) where the HTTP interface program is listening for client connection requests. After that, it creates a `PrintWriter` object to send data over a socket connection to a web server. It also creates a `BufferedReader` object to read the text sent by a server back to a client. It receives the text sent back to it by a server and prints the text out. The `httpServer` class loops on the server and `accept` call waiting for a client connections and creates an instance of the `ClientWorker` class for each client connection it accepts.

Therefore, there are two aims:

1. To make a client multi-threaded, so that it runs the user interface with one thread and handles the I/O with a server in a separate thread.

2. To make a server multi-threaded.

We have handled the user interface with one thread. A second thread listens on a well-known port for client TCP connections. A third thread is started when a client connection is made to handle that client's requests. After launching the thread to handle a client's connection, a server continues listening for additional client connections. If another connection comes in while an earlier client is being served, a fourth thread is started to handle the new client's requests, and so on. Thus, a server includes at least three threads but it may start additional threads for multiple simultaneous connections.

To simplify this programming task, the system is developed the code of HTTP interface mechanism in two stages. In the first stage, we have written a multi-threaded program that simply listens, intercepts, and analyse to display the contents of the HTTP request for verification purposes at the request level. After

this program is running properly, we forward the request to a web server for generating an appropriate response. We have next developed another multi-threaded program that listens, intercepts, and analyses before displaying the contents of the HTTP response for verification purposes at the response level. We forward the response to end user.

The end user can test the TCP connection to a target web server from his/her a web browser where the serving through the standard port 80 so that it is not important to specify this port number within the URL in a web browser. For example, if a server machine name or the the IP address of a server machine is 192.168.10.1, a server is listening to port 80, and we need to retrieve the web file `test.html`, then it is important to specify the following URL within a web browser:

$$http://192.168.10.1/test.html$$

When a web server encounters an error, it sends a response message with the appropriate HTML source so that the error information is displayed in a web browser window.

HTTP interface prototype has contained 8 classes and we have summarised as follows:

1. `HTTPServer` class: extends from thread class. Therefore, two separate classes are defined in the `HTTPServer` class including `HTTPServerWorkerReq`, and `HTTPServerWorkerRes`. The structure of our own `HTTPServer` class is shown in Figure 5.4.

2. `HTTPServerWorkerReq` class: a thread class to listen for each requests on port number 80. We have analysed a request (we have analysed the information in the header lines such as name, size, type of requested web resource and the name of web server) for verification purposes at the request level and forward it to a web sever.

3. `HTTPServerWorkerRes` class: a thread class to listen for each response on port number is 8081.

4. `dbConnection` class: is defined to connect with DBMS Microsoft Access 2007 for storing and retrieving the details for every web content.

5. `Singature` class: is defined to compute the MAC value using SHA-1 hash function and a private key for every web content.

6. `HTTPLog` class: is defined to report the result of the integrity verification process.

7. `originalObject` class: is defined to connect the `dbConnection` class and `Singature` class.

8. `Main` class: is defined to make the main socket and invoke `HTTPServer` class.

There are three parts to the response message: the status line, the response headers and the entity body. The status line and response headers are terminated by the character sequence `CRLF`. In the case of a request for a nonexistent file, we return "404 Not Found" in the status line of the response message, and include an error message in the form of an HTML document in the entity body.

5.2.4 Registry and Integrity Verification using SHA-1 Checksums

A basic function performed by the integrity verifier component is checking that the requested web resources returned by application server match. To improve the efficiency of this process, we use SHA-1 checksum to compute each web resource served. This checksum is cryptographically strong (74; 88; 89; 110): given current computing technologies, the production of a fake web resource that matches a given SHA-1 checksum is currently impossible.

When comparing content of requested or responded web resources, we need to compute the SHA-1 checksum and compare it with the original one. The integrity verifier component compares the checksums and ensures that they match; if so, the client receives the correct requested web resources; otherwise, the integrity

```
import java.io.* ;
import java.net.* ;
import java.util.* ;

public class HTTPServer extends Thread
  {
  public HTTPServer(String wsip, int wsp, int clp)
     {   ....    }

 public void run(){

    while(true){
          try{  ...

  HTTPServerWorkerReq httpwswReq = new
                                HTTPServerWorkerReq(clSocket, wsSocket);
  httpwswReq.start();

  HTTPServerWorkerRes httpwswRes = new
                               HTTPServerWorkerRes(clSocket, wsSocket);
  httpwswRes.start();
   ....
         }
          catch(IOException ioe){
                                HTTPUtil.log(ioe.toString());
                                }
        }
    }
}
```

Figure 5.4: Structure of a HTTPServer Class

verifier forwards this web resource to recovery component to recover it and send it back to integrity verifier for further process.

It should be noted that the current recommendation for newly designed applications is SHA-256 because SHA-1 may be broken (115). However, we have continued to use SHA-1, because:

- SHA1 has been tested in many applications and can be implemented using Java, C#, and C++; while the SHA-256 has not been tested yet (12; 74; 110).
- Although a variety of hash functions are available, MD5 and SHA-1 are in wide use (88; 89).

Readers should note this thesis focuses on an integrity system and does not focus on developing or deploying a new hashing algorithm.

5.3 Testing Strategy

The aim of this testing strategy is to tackle the central points of this thesis: ***How does WCVR system assist in survivability of server-side static and dynamic web content?*** and ***How does WCVR system meet the performance objective in case of static web content and in case of dynamic web content?*** This chapter has described a number of experiments designed to explore these questions.

To evaluate the proposed WCVR system, we have suggested two kinds of testing strategy including security testing strategy and performance testing strategy. Therefore, five experiments have been conducted to find out the behaviour of WCVR system compared with the existing systems such as DSSA and without any verification system in two trends: security (detection and recovery) and performance.

5.3.1 Considerations

Theoretically, the WCVR system can provide a tamper detection and recovery to server-side static and dynamic web content. The WCVR is not confined to a specific web server and web application and it does not involve additional resources and platforms. In the testing design, we have provided JSP web application to generate dynamic web content on Apache Tomcat web server because we have used the powerful features of Java Servlets and Filters. Apache Tomcat service supports the Servlets and Filters technologies but Microsft IIS does not. Currently, there are some commercial products such as `JspISAPI`[3] to run the JSP features on Microsfot IIS which supports Active Server Page (ASP) to generate dynamic web content. Therefore, the proposed WCVR can be scalable to various web servers, platforms, and operating systems.

Note that our response hashing mechanism is implemented by Servlets and Filters in Java for powerful features of Servlets and Filters technologies as explained in Sections 5.1 and 5.2.2.

We believe the WCVR system might offer advantages over the existing systems that the WCVR system can verify, and recover the dynamic web server content as well without the need to restructure the existing web applications (see Section 4.4).

5.3.2 Formal Experimental Statement

The WCVR system can ensure server-side static and dynamic web content survivability (tamper detection, and recovery).

This statement has been tested for two trends: security and performance. We have identified five questions to be answered by this thesis:

1. How does WCVR system provide tamper detection and recovery in the

[3]http://www.neurospeech.com/Products/JspISAPI.aspx

server-side static and dynamic web content on Apache Tomcat web server over wired client-server network?

2. How does WCVR system provide tamper detection and recovery in the server-side static web content on Microsoft IIS web server over wired client-server network?

3. How does WCVR system meet the performance objective compared with DSSA mechanism on IIS web server in the case of static web content?

4. How does WCVR system meet the performance objective compared with DSSA mechanism on Tomcat web server in thecase of static web content?

5. How does WCVR system meet the performance objective compared without any verification system or mechanism (on Tomcat web server) in the case of dynamic web content?

To answer these questions, we have designed the following testing strategy.

5.3.3 Security Testing Strategy

We have tested the proposed WCVR system to detect and recover the server-side web content from the tampering attacks. In other words, we have tackled this question (***How does WCVR system provide tamper detection, and recovery in the server-side static and dynamic web content on Apache Tomcat and Microsoft IIS web servers over wired client-server network?***) by conducting two experiments with taking into account the following Table 5.1:

Therefore, we have carried out two experiments to test the security of the proposed WCVR system:

1. Experiment 1: To investigate the tamper detection and recovery in a server-side static and dynamic web content on Apache Tomcat web server

2. Experiment 2: To investigate the tamper detection and recovery in a server-side static web content on Microsoft IIS web server.

Type of Server-side Web Content	Web Servers	
	Tomcat Web Server	IIS Web Server
Static Web Content	☑	☑
Dynamic Web Content	☑	☒

Table 5.1: Table of the used web servers in the experimental design

It should be noted that we have not carried out an experiment to investigate the tamper detection and recovery in a server-side dynamic web content on IIS web server because the proposed response hashing mechanism is implemented by Java Servlet and Filters and consequently, the IIS web server does not support Java Servlet and Filters.

5.3.3.1 Network Layout

We have constructed a local are a network consisting of two HP server machines: web server and database server, four client machines (2GHZ CEN 1GB RAM OS Windows XP), two routers (CISCO 2800 series) and two switches (Catalyst 2960 series). The Configuration of IP addresses is static for every machine in this LAN.

The network has been used to test the proposed system for security and performance objectives. The Figure 5.5 shows the schematic of the network layout.

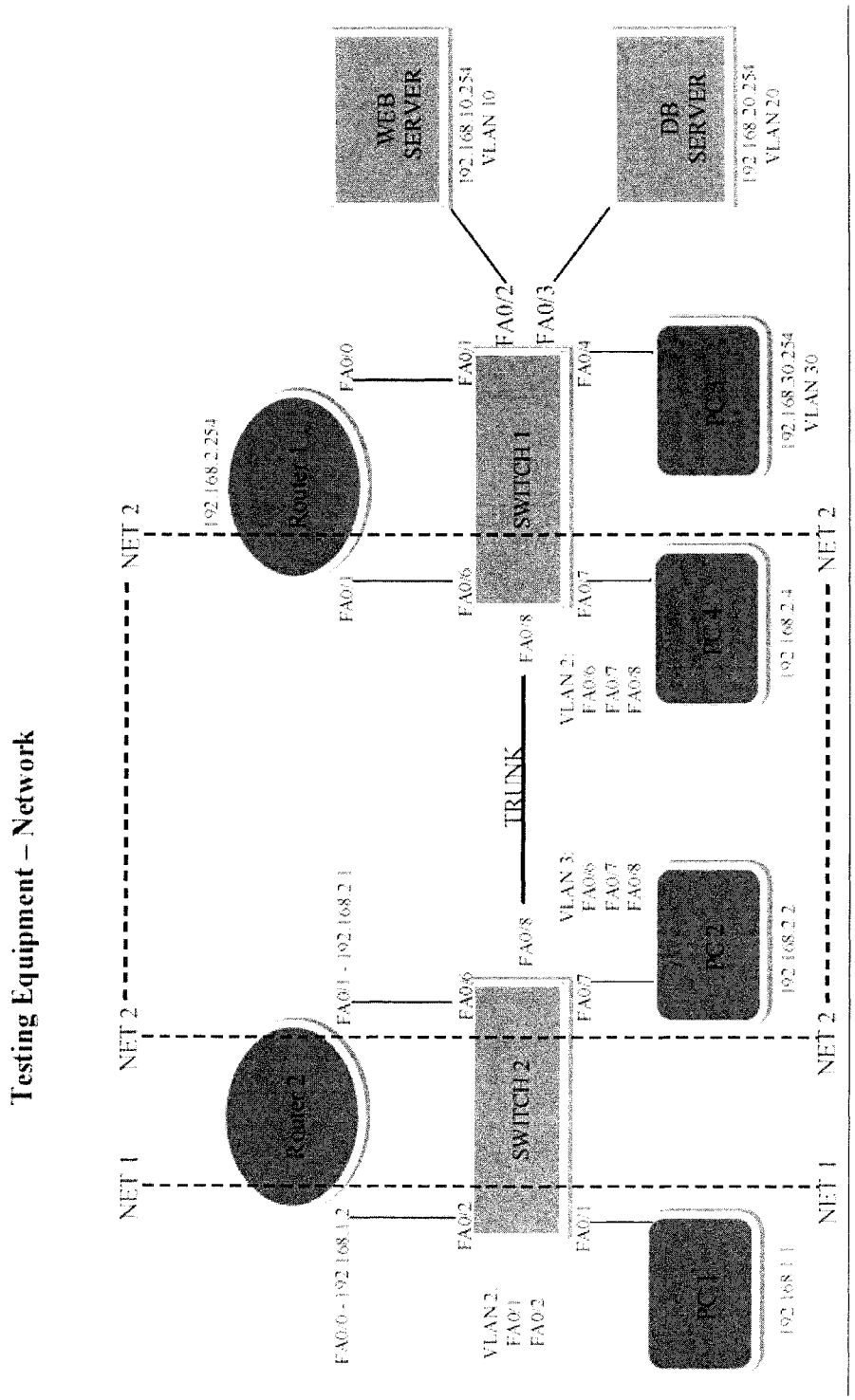


Figure 5.5: The schematic of the network layout for testing purposes.

5.3.3.2 Experiment 1

Experiment 1: Tamper detection, and recovery in the server-side static and dynamic web content using the WCVR system on Apache Tomcat web server over local network.

Participants: Three undergraduate computing students.

Objective:

1. To verify the server-side static and dynamic web content on Tomcat web server using the WCVR system.
2. To recover the server-side static and dynamic web content on Tomcat web server using WCVR system.
3. To illustrate how such attacks can be detected on a Tomcat web server using WCVR system.
4. To illustrate how the modified static and dynamic web content can be recovered on a Tomcat web server using WCVR system.

OS component: Windows server 2003 for server machine, and any windows-type for client's machines.

Other component: Tomcat Apache Server (v5.0.28).

Network Architecture: Server machines, two switches, two routers, and four client machines (2GHZ CEN 1GB RAM OS Windows XP) over wired network.

Tools (Required tools for the experiment):

1. WCVR system.
2. Apache Tomcat web server.
3. Request capture tool to automatically launch various tampering attacks to web content.

4. Paint tool to manually tamper any static image.
5. Notepad tool to manually tamper any static web content.
6. MS Internet Explorer (IE) or Mozilla Firefox.

Data:

1. Given Borland JBuilder JSP shopping cart[4] and UK Hillside Primary school web site[5]

Setup:

1. Construct LAN network which contains a server machine, three client's machines and two routers using TCP protocol.
2. Pick every client machine and check connectivity with server.
3. Install Apache Tomcat service on a server. This service will be listening on every request and response with port number 8081 where the default port number is 80.
4. Make two copies of the target web site. One on a web server and one in secured directory for recovery purposes.

Procedure: The following procedures take place on the server and client sides.

Server-side

1. Run Web Register mechanism.
2. Copy the target web site to Tomcat root.
3. Run Tomcat web server.
4. Run the HTTP interface mechanism.

[4]http://www.borland.com/uk/products/jbuilder/
[5]http://hillside.needham.k12.ma.us/cyberventues/st_proj.html

5. Run the response hashing mechanism.
6. Launch a number of tampering attacks by (i) change manually any web content on the root of Tomcat web server, (ii) run the request capture tool for automatic modification.
7. Repeat step 6.

Client-side

1. Run the IE web browser in a client machine (Start-Programs-Internet Explorer)
2. Enter the following URL (HTTP://192.168.10.1/test.html) in address bar.
3. Check the requested web resources.
4. Is it the original one or the tampered one? If the original file, this mean the WCVR system does detect this tampering attack and recover the requested web resource.
5. repeat step 4.

5.3.3.3 Experiment 2

Experiment 2: Tamper detection, and recovery in the server-side static web content using WCVR system on Microsoft IIS web server.

Participants: Three undergraduate computing students.

Objective:

1. To verify the server-side static web content on IIS web server using the WCVR system.
2. To recover the server-side static web content on IIS web server using WCVR system.

3. To illustrate how such attacks can be detected on IIS web server using WCVR system.

4. To illustrate how the modified static web content can be recovered on on IIS web server using WCVR system.

OS component: Windows server 2003 for server machine, and any windows-type for client's machines.

Other component: IIS web server (v6.0).

Network Architecture: Server machines, two switches, two routers, and four client machines (2GHZ CEN 1GB RAM OS Windows XP) over wired network.

Tools (Required tools for the experiment):

1. WCVR system consisting of Web Register mechanism and HTTP Interface mechanism.

2. IIS web server.

3. Request capture tool to automatically launch various tampering attacks to static web content.

4. Paint tool to manually tamper any static image.

5. Notepad tool to manually tamper any static web content.

6. MS Internet Explorer (IE) or Mozilla Firefox.

Data:

1. Given UK Hillside Primary school web site.

Setup:

1. Construct LAN network which contains a server machine, three client's machines and two routers using TCP protocol.

2. Pick every client machine and check connectivity with server.
3. Install IIS service on a server. This service will be listening on every request and response with port number 80.
4. Make two copies of the target web site. One on a web server and one in secured directory for recovery purposes.

Procedure: The following procedures take place on the server and client sides.

Server-side

1. Run Web Register mechanism.
2. Copy the target web site to the root (`wwwroot`) of IIS folder.
3. Run IIS service.
4. Run the HTTP interface mechanism.
5. Launch a number of tampering attacks by (i) change manually any static web content on the root of IIS web server, or (ii) run the request capture tool for automatic modification.
6. Repeat step 5.

Client-side

1. Run the IE web browser in a client machine (Start-Programs-Internet Explorer)
2. Enter the following URL (HTTP://192.168.10.1/test.html) in address bar.
3. Check the requested web resources.
4. Is it the original one or the tampered one? If the original file, this mean the WCVR system does detect this tampering attack and recover the requested web resource.
5. repeat step 4.

5.3.4 Performance Testing Strategy

To test the performance of the WCVR system, we have used Neoload[6] application to find out the end-to-end performance measurement over a wired network. In case of static web content, there is a comparison between the proposed WCVR system with DSSA mechanism on the both web servers: Tomcat and IIS.

It should be noted that in the case of dynamic web content, the comparison focuses on the differences between using the WCVR and not using the WCVR - currently there is no verification system able to detect and recover dynamic web content.

A load test can be used to test an application's robustness and performance, as well as its hardware and bandwidth capacities (129). Therefore, we used the Neoload application which is a stress and load testing tool to:

1. test a web site's vulnerability to crashing under load.
2. check response times under the predicted load.
3. determine the number of simultaneous users supported by the application.
4. define hardware and bandwidth requirements.

The NeoLoad application has a number of features that motivate us to use it in our testing as follows:

1. Recording HTTP traffic between browser and server
2. Defining virtual user behavior and setting scenario parameters such as load policy (constant, ramp-up, peak) or the number of virtual users to be simulated in the test.
3. Setting performance monitors (CPU, memory, disk usage and others) for your servers.

[6]http://www.neotys.com/

4. Monitoring the ongoing test with the aid of real-time graphs and states.
5. Obtaining a summary of the test and then examining its details using the graphs and statistical tables.

5.3.4.1 Experiment 3

Experiment 3: To measure end-to-end performance of static web content on Microsoft IIS.

Participants: Three undergraduate computing students.

Objectives:

1. To find out the response time (i.e. the running time between a request and the reception of its answer including web server response time, communication response time, and browser response time) for the following two cases:
   (a) With the DSSA system.
   (b) With the WCVR system.
2. To compare the results in accordance to the response time and overhead times among them and then evaluate the performance of WCVR system.
3. To find out the statistics summary such as average and standard deviation of response time for request and page, average throughout, and number of hits.

OS component: Windows server 2003 for server machine, and any windows-type for client's machines.

Other component: IIS.

Network Architecture: Server machines, two switches, two routers, and four client machines (2GHZ CEN 1GB RAM OS Windows XP) over wired network.

Tools(Required tools for the experiment):

1. WCVR system.
2. IIS web server.
3. Request capture tool to automatically launch various tampering attacks to static web content.
4. Paint tool to manually tamper any static image.
5. Notepad tool to manually tamper any static web content.
6. MS Internet Explorer (IE) or Mozilla Firefox.
7. DSSA mechanism.
8. Neoload application.

Data:

1. Given UK Hillside Primary school web site.

Setup:

1. Install IIS service on a server. This service will be listening on every request and response with port 80.
2. Construct LAN network which contains a server machine, three client's machines and two routers using TCP protocol.
3. Pick every client machine and check connectivity with server.
4. Make two copies of the target web site. One on a web server and one in secured directory for recovery purposes.
5. Install DSSA mechanism.
6. install Neoload application.

Procedure: The following procedures take place on the server and client sides.
Case (a)

Server-side

1. Run Web Register mechanism.
2. Run IIS service.
3. Run the DSSA system.

Client-side

1. Run Neoload tool.
2. Request the following URL (HTTP://192.168.10.1/test.html).
3. Generate a separate report.

Case (b)

Server-side

1. Run Web Register mechanism.
2. Run IIS service.
3. Run the HTTP interface mechanism.

Client-side

1. Run Neoload tool.
2. Request the following URL (HTTP://192.168.10.1/test.html).
3. Generate a separate report.

5.3.4.2 Experiment 4

Experiment 4: To measure end-to-end performance of static web content on Apache Tomcat.

Participants: Three undergraduate computing students.

Objectives:

1. To find out the response time for the following two cases:
   (a) With the WCVR system.
   (b) With the DSSA system.
2. To compare the results in accordance to the response time and overhead times among them and then evaluate the performance of WCVR system.
3. To find out the statistics summary such as average and standard deviation of response time, number of hits, average throughput, and number of errors and maximum and minimum user load.

OS component: Windows server 2003 for server machine, and any windows-type for client's machines.

Other component: Apache Tomcat web server.

Network Architecture: Server machines, two switches, two routers, and four client machines (2GHZ CEN 1GB RAM OS Windows XP) over wired network.

Tools(Required tools for the experiment):

1. WCVR system consisting of Web Register mechanism and HTTP Interface mechanism.
2. Apache tomcat web server.
3. Request capture tool to automatically launch various tampering attacks to static web content.

4. Paint tool to manually tamper any static image.
5. Notepad tool to manually tamper any static web content.
6. MS Internet Explorer (IE) or Mozilla Firefox.
7. DSSA system.
8. Neoload application.

Data:

1. UK Hillside Primary school web site.

Setup:

1. Install Apache Tomcat service on a server. This service will be listening on every request and response with port number 8081 where the default port number is 80.
2. Construct LAN network which contains a server machine, three client's machines and two routers using TCP protocol.
3. Pick every client machine and check connectivity with server.
4. Make two copies of the target web site. One on a web server and one in a secured directory for recovery purposes.
5. Install DSSA mechanism.
6. install Neoload tool.

Procedure: The following procedures take place on the server and client sides.

Case (a)

Server-side

1. Run Web Register mechanism.

2. Run Tomcat web server.
3. Run the HTTP interface mechanism.

Client-side

1. Run Neoload tool.
2. Request the following URL (HTTP://192.168.10.1/test.html).
3. Generate a separate report.

Case (b)

Server-side

1. Run Web Register mechanism.
2. Run Tomcat web server.
3. Run the DSSA mechanism.

Client-side

1. Run Neoload tool.
2. Request the following URL (HTTP://192.168.10.1/test.html).
3. Generate a separate report.

5.3.4.3 Experiment 5

Experiment 5: To measure end-to-end performance of dynamic web content on Apache Tomcat.

Participants: Three undergraduate computing students.

Objectives:

1. To find out the response time for the following two cases:
   (a) Without any mechanism or system for tamper detection and recovery.
   (b) With the WCVR system.
2. To compare the results in accordance to the response time and overhead times among them and then evaluate the performance of WCVR system.
3. To find out the statistics summary such as average and standard deviation of response time, number of hist, average throughout, and number of errors and maximum and minimum user load.

OS component: Windows server 2003 for server machine, and any windows-type for client's machines.

Other component: Tomcat.

Network Architecture: Server machines, two switches, two routers, and four client machines (2GHZ CEN 1GB RAM OS Windows XP) over wired network.

Tools(Required tools for the experiment):

1. WCVR system consisting of Web Register mechanism, HTTP Interface mechanism and Response Hashing Calculator mechanism.
2. Apache tomcat web server.
3. Request capture tool to automatically launch various tampering attacks to static web content.
4. Paint tool to manually tamper any static image.
5. Notepad tool to manually tamper any static web content.
6. MS Internet Explorer (IE) or Mozilla Firefox.
7. Tomcat service.
8. javac command.

9. Neoload tool.

Data:

1. Borland JBuilder JSP shopping cart.

Setup:

1. Install Apache Tomcat service on a server. This service will be listening on every request and response with port number 8081 where the default port number is 80.
2. Construct LAN network which contains a server machine, three client's machines and two routers using TCP protocol.
3. Pick every client machine and check connectivity with server.
4. Make two copies of the target web site. One on a web server and one in secured directory for recovery purposes.
5. Install Neoload tool.

Procedure: The following procedures take place on the server and client sides.

Case (a)

Server-side

1. Run Apache Tomcat web server.

Client-side

1. Run Neoload tool.
2. Request the following URL (HTTP://192.168.10.1:8081/test.html).
3. Generate a separate report.

Case (b)

Server-side

1. Run Web Register mechanism.
2. Copy the java classes to WEB-INF of Tomcat root.
3. Run Tomcat web server.
4. Run the HTTP interface mechanism.
5. Run the response hashing mechanism.

Client-side

1. Run Neoload tool.
2. Request the following URL (HTTP://192.168.10.1/test.html).
3. Generate a separate report.

5.4 Initial Testing

We carried out a pilot study to evaluate the reliability and effectiveness of our proposed system. The objective of the initial test is to evaluate the proposed approach in term of both reliability and performance of the tamper detection and recovery processes. The reliability of the approach is its ability to correctly detect the tampering attacks and perform recovery if any tampering has been detected.

In this Section, we detail the phase of the evaluation process. The reliability is evaluated by launching manually tampering attacks against the generated dynamic and static web content on IIS and Tomcat web servers.

In the pilot study, we carried out these experiments to illustrate the two main objectives:

1. Security Objective: ***How does WCVR system provide tamper detection, and recovery in server-side static and dynamic web content on Apache Tomcat and IIS web servers?***

2. Performance Objective: ***How does WCVR system meet the performance objective in case of static web content and in case of dynamic web content?***

We carried out experiments 1e, and 2 to meet the security objective, whereas, we carried out experiments 3, 4, and 5 to meet the performance objective.

Experiment 1 was carried out on MS Windows environment and Apache Tomcat web server. After running this experiment we had manually and/or automatically launched 45 tampering attacks (such as visualisation spoofing attack, textual spoofing attack, tampering code manipulation tampering object manipulation, and defacement of web page), and as a result, we obtained a set of results as shown in Table 5.2. This results has shown that that our proposed WCVR system provides tamper detection, and recovery in server-side static and dynamic web content. Table 5.2 illustrates how such attacks can be detected using the proposed WCVR system.

N	Http Request	Detection	Recovery
1	/StartHere.jsp	YES	YES
2	/images/greenpaper.gif	YES	YES
3	/images/dollar.gif	YES	YES
4	/books.gif	YES	YES
5	/favicon.ico	YES	YES
6	/Store.jsp	YES	YES
7	/images/rarrow.gif	YES	YES
8	/Cart.jsp	YES	YES
9	/images/larrow.gif	YES	YES
10	/Store.jsp	YES	YES
11	/Cart.jsp	YES	YES
12	/Store.jsp	YES	YES
13	/Cart.jsp	YES	YES
14	/Store.jsp	YES	YES
15	/Cart.jsp	YES	YES
16	/Store.jsp	YES	YES
17	/Cart.jsp	YES	YES
18	/Store.jsp	YES	YES
19	/Cart.jsp	YES	YES
20	/Store.jsp	YES	YES
21	/Cart.jsp	YES	YES
22	/Purchase.jsp	YES	YES
23	/ThankYou.jsp	YES	YES
24	/TreeDiagrams/cyberventues/Hachett_trees07/trees.htm	YES	YES
25	/TreeDiagrams/cyberventues/Hachett_trees07/arbres-07.gif	YES	YES
26	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/natal	YES	YES
27	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/natal	YES	YES
28	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/denni	YES	YES
29	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/denni	YES	YES
30	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/kater	YES	YES
31	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/kater	YES	YES
32	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/glenn	YES	YES
33	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/glenn	YES	YES
34	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/vali.	YES	YES
35	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/vali_	YES	YES
36	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/skye.	YES	YES
37	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/skye_	YES	YES
38	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/kenda	YES	YES
39	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/kenda	YES	YES
40	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/andre	YES	YES
41	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/andre	YES	YES

Table 5.2: Table of the static and dynamic web content in Tomcat web server

Figure 5.6 shows a (partial) list of measurements for the original hash values of web contents which is shown in Table 5.2. The measurements in Figure 5.6 have been taken by the web register mechanism. Figure 5.7 shows the corresponding list of the same web contents that were compromised. The entries in Figure 5.7 have illustrated that after the attack, the signature of the `arbres-07.gif` was different, indicating that the attack replaced the original `arbres-07.gif` with a faked copy.

Experiment 2 was carried out in MS Windows environment and IIS web server. After running this experiment we had manually and/or automatically launched 45 tampering attacks, and as a result, we obtained a set of results as

shown in Table 5.3. This results has shown that that our proposed WCVR system provides tamper detection and recovery in server-side static web content. Table 5.3 illustrates how such attacks can be detected using the WCVR system.

```
#001: 6FD08C991E77FD867C96D86E64C1095620E69D29 vivian.htm
#002: 150E93F8C29EC534178DA0647251807BDC12BFF2 vivian_1.GIF
#003: 0DC2F919A3BBBC6554A462AD80AE57679B9C6969 Thumbs.db
#004: 1E7707F952FCBF9627076FAE387C1E6685FA6192 trees.htm
...
#040: 0B81CE4C28596973C97ED109C3A44DAF8D41F105 arbres-07.gif
...
```

Figure 5.6: A list of measurements for the original hash values of web contents

```
...
#040: EB81CE4C77796973C97ED109C3A44DAF8D41FDE8 arbres-07.gif
...
#400: 1FB659F0535DAAD725FFB79C6C0458C75685128A  andrew.html
...
```

Figure 5.7: Detecting tampering attacks

N	Http Request	Detection	Recovery
1	/TreeDiagrams/cyberventues/Hachett_trees07/trees.htm	YES	YES
2	/TreeDiagrams/cyberventues/Hachett_trees07/arbres-07.gif	YES	YES
3	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/natal	YES	YES
4	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/natal	YES	YES
5	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/natal	YES	YES
6	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/kiana	YES	YES
7	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/kiana	YES	YES
8	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/kiana	YES	YES
9	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/denni	YES	YES
10	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/denni	YES	YES
11	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/denni	YES	YES
12	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/elan.	YES	YES
13	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/elan.	YES	YES
14	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/elan_	YES	YES
15	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/kater	YES	YES
16	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/kater	YES	YES
17	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/kater	YES	YES
18	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/iness	YES	YES
19	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/iness	YES	YES
20	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/iness	YES	YES
21	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/nia.h	YES	YES
22	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/nia.h	YES	YES
23	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/nia_f	YES	YES
24	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/danny	YES	YES
25	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/danny	YES	YES
26	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/danny	YES	YES
27	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/glenn	YES	YES
28	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/glenn	YES	YES
29	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/glenn	YES	YES
30	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/kenda	YES	YES
31	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/kenda	YES	YES
32	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/kenda	YES	YES
33	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/kenda	YES	YES
34	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/vivia	YES	YES
35	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/vivia	YES	YES
36	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/andre	YES	YES
37	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/andre	YES	YES
38	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/vali.	YES	YES
39	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/vali_	YES	YES
40	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/micha	YES	YES
41	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/micha	YES	YES
42	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/skye.	YES	YES
43	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/skye_	YES	YES
44	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/serge	YES	YES
45	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/serge	YES	YES
46	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/jared	YES	YES
47	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/jared	YES	YES
48	/TreeDiagrams/cyberventues/Hachett_trees07/amandahtml/ben.h	YES	YES

Table 5.3: Table of the static web content in IIS web server

Experiment 3 and 4 were individually carried out using MS Windows environment (i.e. MS Windows XP Professional for client machines and MS Windows Server 2003) with IIS and Tomcat web servers. Experiment 3 is designed to show end-to-end performance (i.e. verification of the integrity of server-side static web content, recovery response if a web content has been tampered with, network speed, web server response and web browser response) via (i) the proposed WCVR system and (ii) the existing DSSA system on IIS web server. Experiment 4 is designed to show end-to-end performance via (i) the proposed WCVR system and (ii) the existing DSSA system on Tomcat web server.

In this pilot study, the duration of the test was almost exactly 5 minutes where the generating a number of virtual users was almost 5 that increased throughout the test. The virtual users were connecting at 100Mbps through a local network.

We have summarised the results in Table 5.4. All these measurements were performed from the client point of view (i.e. all durations show the time between a request and the reception of its answer). Each row in the table displays the average response time (request) maximum response time (request), and minimum response time (request) in seconds of all requests during the test and average page response time for all pages where each page may contain a number of requests. Note that the average response time is the mean-time necessary to process a request by each web server when the proxy, browser and the WCVR system are active. The activation of the WCVR implies that HTTP request-response communication has changed. The activation of the of the recovery component implies that the server-side static and/or dynamic web content has been tampered with. The communications (network response time) are parts of the measured duration.

A comparison of the WCVR and DSSA systems indicates that the end-to-end performance of the WCVR (on IIS and Tomcat web servers) is better than that on the DSSA as illustrated in Table 5.4. The average response time (request) through the WCVR was 0.574 seconds on IIS and was 1.05 seconds on Tomcat, while the average response time (request) through DSSA was 0.648 seconds on IIS and was 2.42 seconds on Tomcat.

System	Web Server	Graph Min (request)	Average response time (request)	Graph Max (request)	Median (request)	Average 90% (request)	Std. Deviation (request)	Average page response time
DSSA	IIS	0.052	0.648	4.06	0.146	0.501	1.07	0.942
WCVR	IIS	0.075	0.574	4.03	0.092	0.414	1.12	0.54
DSSA	Tomcat	0.153	2.42	4.03	3.96	2.41	1.86	3.03
WCVR	Tomcat	0.078	1.05	4.04	0.099	0.916	1.65	1.05

Table 5.4: Comparison between the response times through DSSA and WCVR systems, in seconds, of all requests during the test on IIS and Tomcat web servers.

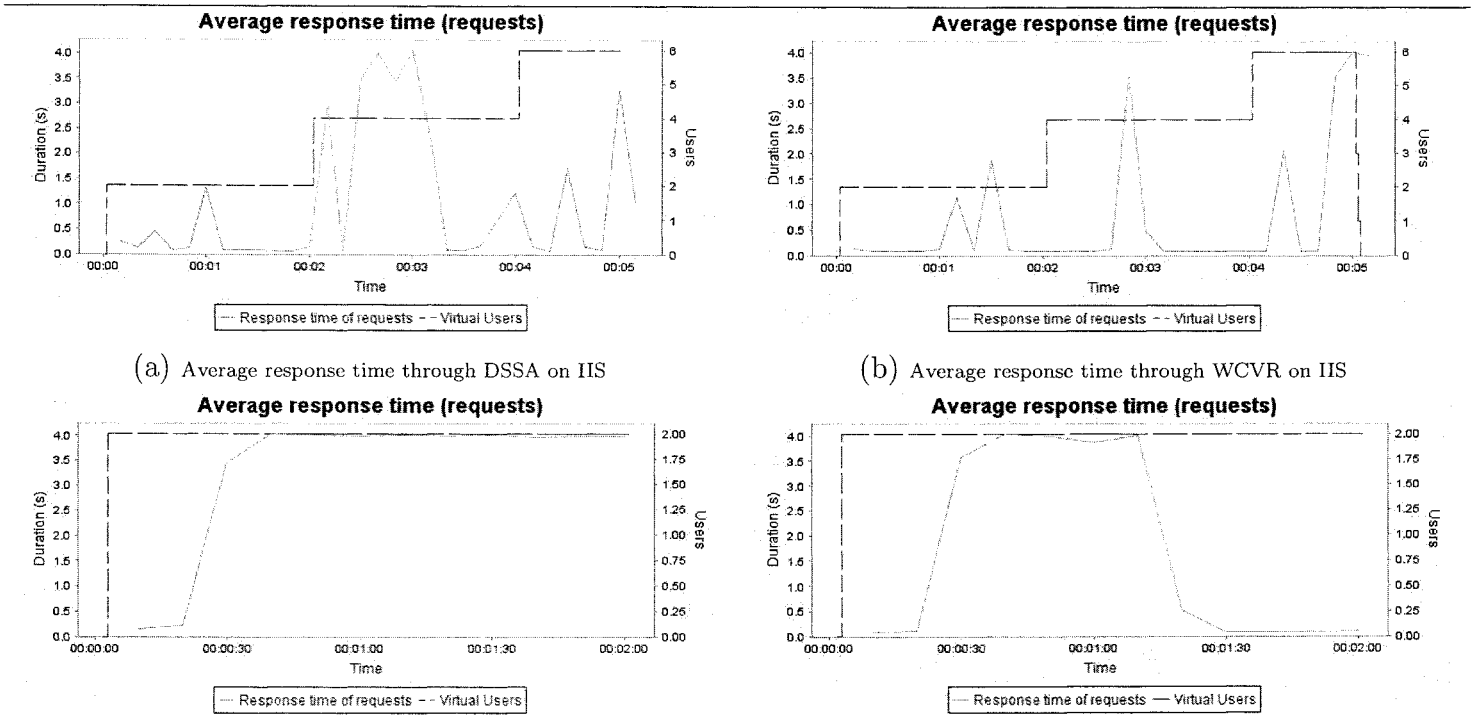
The running time increased close to a linear state as the number of users increased as shown in Figure 5.8. The curves in Figure 5.8 show the average response time in seconds of all requests and the number of virtual users as the vertical axis during the ramp-up (increasing) load test. Note that we used the Neoload product to measure the performance, where the Neoload generates number of virtual users after recording all requests and responses during the test.

The WCVR is less costly in performance terms when verifying the server-side static web content against tampering attacks. This is understandable, because we utilised the WCVR system by formulating a new hashing strategy. It is anticipated that our system can give better results for security and performance compared with other existing systems. In this study, we can see the overhead is acceptable and is adapted to a real-time use.

Figure 5.9 displays the percentage of pages that were performed within a given time range. These graphs help determine the percentage of pages that meet a performance objective. For example, graph (a) in Figure 5.9 presents the distribution of response page time for DSSA on IIS web server within 4 seconds. It indicates that 90% of the requested pages had a response time under 3.5 seconds. Where as graph (b) in Figure 5.9 presents the distribution of response page time for WCVR on IIS web server within 4 seconds. It indicates that 90% of the requested pages had a response time under 2.0 seconds. This suggests that the WCVR takes less response time as a percentage to verify a static web content on

IIS than DSSA.

Graph (c) in Figure 5.9 presents the distribution of response page time for DSSA on Tomcat web server over 4 seconds. It demonstrates that about 25% of the requested pages had a response time under 0.5 second. Graph (d) in Figure 5.9 presents the distribution of response page time for WCVR on Tomcat web server within 4 seconds. This indicates that 70% of the requested pages had a response time under 0.5 seconds. This suggests that the WCVR takes much less response time as percentage to verify a static web content on Tomcat than DSSA. In this study, the WCVR (when compared with DSSA) satisfies the performance objective as seen in Section 5.4.



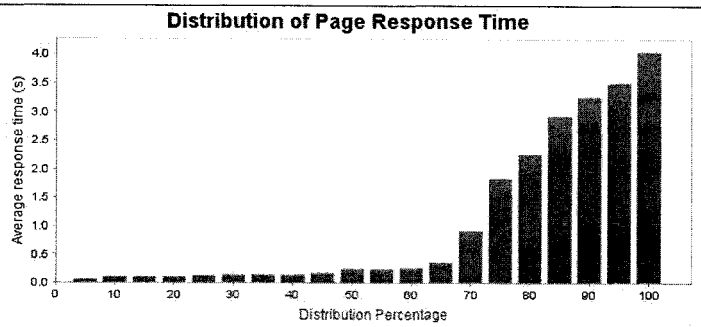
(a) Average response time through DSSA on IIS

(b) Average response time through WCVR on IIS

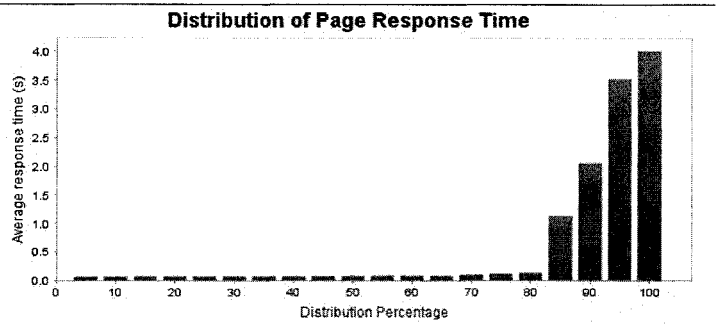
(c) Average response time through DSSA on Tomcat

(d) Average response time through WCVR on Tomcat

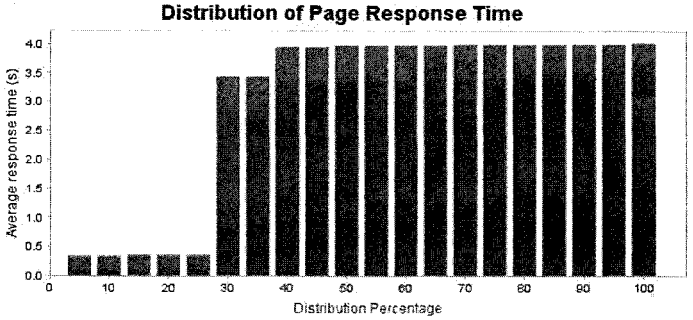
Figure 5.8: Static web content: Graphs for response time (request) curves through DSSA and the WCVR on IIS and Tomcat servers, in seconds, of all requests during the test.



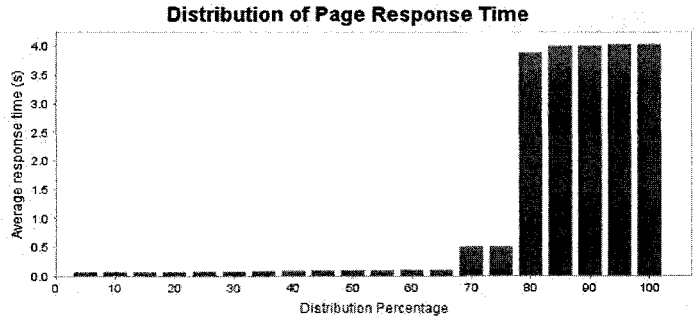
(a) The distribution of page response time for DSSA on IIS.



(b) The distribution of page response time WCVR on IIS.



(c) The distribution of page response time for DSSA on Tomcat.



(d) The distribution of page response time for WCVR on Tomcat.

Figure 5.9: Displays the distribution of page response time, in seconds, of all pages during the test.

Experiment 5 was carried out in a MS Windows environment (i.e. MS Windows XP Professional for clients machines and MS Windows Server 2003) using a Tomcat web server. This experiment is designed to show end-to-end performance (i.e. verification of the integrity of server-side dynamic web content, recovery response if a web content has been tampered with, network speed, web server response, and web browser response) via (i) the proposed WCVR system and (ii) without any verification system on Tomcat web server. In this study, the response times for Scenario A (without any mechanism or system for tamper detection and recovery) and Scenario B (with the WCVR system) were collected.

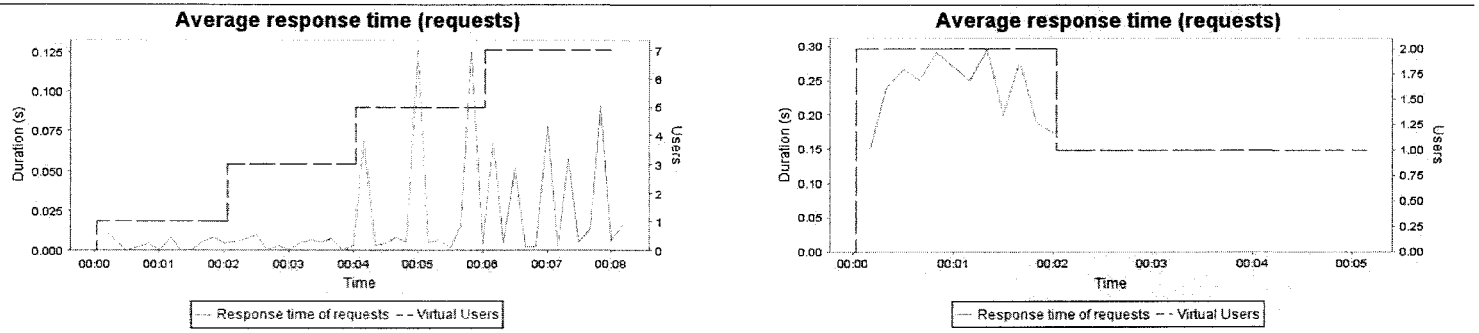
Table 5.5 shows the statistics summary such as the HTTP requests, the minimum response time, average response time, and maximum response time. In Scenario A, the minimum request response time was 0.01 second, average response time as 0.021 seconds, maximum response time was 0.126 seconds, standard deviation was 0.031 seconds. whereas in Scenario B, the minimum request response time was 0.149 second, average response time as 0.212 seconds, maximum response time was 0.297 seconds, standard deviation was 0.051 seconds.

System	Web Server	Graph Min (request)	Average Response Time (request)	Graph Max (request)	Standard Deviation
Without any verification system	Tomcat	<0,01	0.021	0.126	0.031
WCVR	Tomcat	0.149	0.212	0.297	0.051

Table 5.5: Comparison between the response times without verification system and WCVR systems, in seconds, of all requests during the test on Tomcat web server.

As shown in Table 5.5, the end-to-end performance of WCVR system is relatively acceptable because the overhead of WCVR system on Tomcat web server is minimal in comparison with Scenario A. The average response time (request) through WCVR was 0.212 seconds on Tomcat, whereas the average response time (request) when using without any verification system was 0.021 seconds on Tomcat. The running times of WCVR system is more time than the running times without any verification system.

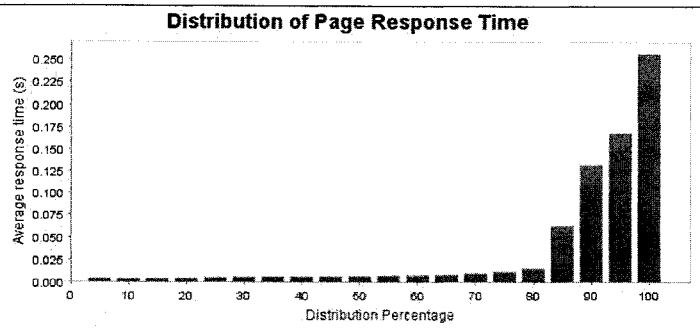
The curve (a) in Figure 5.11 shows the percentage of pages (distribution of page response time)that were performed within a given time range. it indicated that 90% of the pages had a response time under 0.125 seconds. The curve (b) in Figure 5.11 indicates that 90% of the pages had a response time under 0.55 seconds.



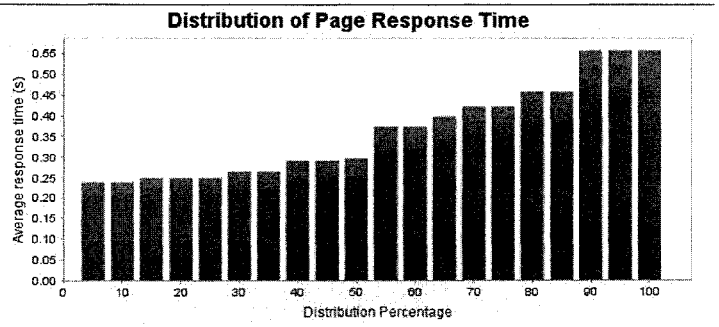
(a) Average response time through without verification mechanism on Tomcat

(b) Average response time through WCVR on Tomcat

Figure 5.10: Dynamic Web Content: Graphs for response time curves through without verification system and the WCVR on Tomcat, in seconds, of all requests during the test.



(a) The distribution of dynamic page response time without verification system on Tomcat.



(b) The distribution of dynamic page response time for WCVR on Tomcat.

Figure 5.11: Displays the percentage of dynamic pages that were performed within a given time range. Theses graphes help determine the percentage of pages that meet a performance objective.

5.5 Conclusion

The WCVR prototype is implemented into three mechanisms: web register, response hashing and HTTP interface mechanism. We carried out a set of experiments to address the security objective (detection, and recovery) and the performance objective. The results of a pilot study have shown that the proposed system (WCVR) provides a high coverage of detection and recovery.

The performance evaluation of a pilot study is limited and does not show a significant enhancement. Thus, our pilot study is not sufficient to evaluate the proposed WCVR system for the following four reasons:

1. The test duration in the pilot study was only 5 minutes. The duration of the test should change depending on the testing goals. The performance information can be obtained for a test that takes a few minutes long. However, we will try to stress a web site/application to know if anything breaks using the WCVR and DSSA. In addition, we will gain more useful performance information and hence we will run the test over a longer period of time.

2. The number of virtual users started with 5 and reached up to 30. The virtual user traffic to load test a web site/application should nearly be to real traffic, and hence we will run the test using a greater number of virtual users.

3. The number of tampering attacks was 45. We will run three experimental studies. Each one will launch 45 tampering attacks. This will be stronger evidence that the WCVR is able to detect and recover tampering attacks.

4. The experimental studies have been conducted by one user.

To achieve more reality in our system evaluation, the next chapter will present further evaluation to the experimental results depending on a number of evaluation criteria such as response times and error rate. The objective of the tests that have been conducted is to evaluate the proposed approach in terms of both

reliability of the tamper detection and recovery processes. The reliability of the approach is its ability to detect correctly the tampering attacks and recover them if any tampering has been detected.

Chapter 6

System Evaluation

6.1 Introduction

In the previous chapter, the implementation of the prototype and design of the experiments have been explained. To test our approach, we implement the WCVR system to verify real-world web applications. We conducted a pilot study to test the reliability and performance of WCVR system and the results obtained appear to demonstrate that the overhead time for the verification process and recovery process are relatively low and that the WCVR system can efficiently and correctly determine if a web content has been compromised.

The findings in (121; 130) have shown how easy and effective for an adversary to automatically find potentially vulnerable web sites. An adversary could probably create a list containing several hundred vulnerable web sites using malicious web content manipulation software run on high-performance servers, a high-bandwidth uplink and several weeks of scanning.

In this chapter, we present five experimental studies to evaluate the reliability and performance of WCVR as follows:

1. Case study - Security objective to evaluate whether the WCVR is able to detect and recover from the tampering attacks.

2. Case study for micro-benchmarking[1] performance to measure offline the performance of the web register mechanism (using SHA-1 and SHA1-extended)
.

3. Case study K to measure the end-to-end performance with (i) the proposed WCVR system, (ii) the existing DSSA system, and (iii) without any verification system.

4. Case study R to measure the end-to-end performance through (i) the proposed WCVR system, (ii) the existing DSSA system and (iii) without any verification system.

5. Case study J to measure the end-to-end performance through (i) the proposed WCVR system, (ii) the existing DSSA system, and (iii) without any verification system.

It should be noted that in each of the above case studies, inputs such as bandwidth, throughput and number of hits, etc. are varied.

6.2 Reflections on Methodology

There are several research methodologies in computing and software engineering areas such as simulation, mathematical or/and logical proof and experimental study (126). Our thesis has used the experimental study to test the hypotheses related to conceptual design and to system reliability and performance.

This methodology has helped to answer the research question which is used to develop the hypothesis verified by the experimental study. The conclusion derived from the experimental results supports the hypothesis as follows:

- By confirming the conceptual approach to find a solution through implementation and testing.

[1]Benchmarking is a process to measure, assess and describe performance of a mechanism or a certain function against agreed criteria. Further details can be found in www.citywestwater.com.au/about/glossary.htm.

- By showing that the performance of WCVR is relatively effective.
- By illustrating how tampering attacks can be discovered, and how tampered web content can be detected and recovered reliably using the WCVR system.

6.3 System Evaluation: Security and Performance

To verify the accuracy and reliability of WCVR in detecting tampering attacks, three computing students at the Computer Forensics lab at Northumbira University picked over five hundred web requests from the (i) UK Hillside Primary school web site[2] and (ii) Borland JBuilder JSP shopping cart[3]. The students identified a potential victim list of target web resources and manually confirmed exploitable flaws in the identified web resources.

In order to evaluate the security objective, the students performed the following two experiments:

- Case study 1 (see Section 6.2.1)
  - Experiment 1: To investigate the tamper detection and recovery in a server-side static and dynamic web content on Apache Tomcat web server.
  - Experiment 2: To investigate the tamper detection and recovery in a server-side static web content on Microsoft IIS web server.

To measure the runtime performance of the web register mechanism using SHA-1 and SHA1-extended, we will present the following case study:

- Case study 2 (see Section 6.2.2)

[2]http://hillside.needham.k12.ma.us/cyberventues/st_proj.html
[3]http://www.borland.com/uk/products/jbuilder/

To evaluate the effectiveness of the WCVR system, the students individually conducted the following experiments:

- Case study 3, 4, and 5 (see Section 6.2.3)
  - Experiment 3: To measure end-to-end performance of static web content on Microsoft IIS.
  - Experiment 4: To measure end-to-end performance of static web content on Apache Tomcat.
  - Experiment 5: To measure end-to-end performance of dynamic web content on Apache Tomcat.

It should be noted that we have not carried out an experiment to investigate the tamper detection and recovery in a server-side dynamic web content on IIS web server because the proposed response hashing mechanism is implemented by Java Servlet and Filters and consequently, the IIS web server does not support Java Servlet and Filters.

Defining the success or failure criteria is a prerequisite to any experimental test. Therefore, before testing a web application, we have defined a number of the acceptable levels for robustness and performance. These criteria are defined in terms of response time as follows:

- Average response time per page (may be different from one page to another).
- Maximum response time per request.
- Minimum response time per request.
- Average response time per request.
- Standard deviation of response time per request.

A web page contains one or more HTTP requests. Wherever web pages or containers are concerned:

- The response time is the time taken to respond to all the requests contained in the page or container.
- The size is the sum of the sizes of all the requests contained in the page or container.
- A page or container is flagged as containing an error if one of its sub-requests contains an error.

6.3.1 Case Study - Security Objective (Detection and Recovery)

As stated in (131), the security objective is difficult to measure. There is no standard accepted methodology by which to evaluate it. To assess how successful the WCVR system is able to detect and recover from the unknown tampering attacks students (K, R, and J) started to run the proposed web register mechanism. This mechanism takes the hashing measurements for every server-side static web content stored in the designated directories of the suggested web sites/applications hosted on Tomcat and IIS web servers.The response hashing mechanism takes the hashing measurement for the generated dynamic web content during online transactions (when a user request a web resource). Over 70 tampering attacks were launched against the designated directories of the suggested web sites hosted on Tomcat and IIS web servers. Measurements were taken after the tampering attacks for integrity verification purposes (see Table 6.1).

Table 6.1 shows a partial list of measurements for the web content and Table 6.2 shows the corresponding list of the same web content that has been compromised by the tampering attacks. The entries in Table 6.2 illustrates that after the attack, the checksum of the requested resource `"/cyberventues/st_proj.html"` is different, indicating that the malicious software replaced the original version. The WCVR system uses the difference in the hash value to detect whether malicious software has replaced the original resource. The verification process checks to see if the web content has been modified since it was used. Based on whether the test passes or fails, the HTTP interface mechanism executes the state protocol to enforce the policy that makes the decision about the next step in the process. If the integrity check passes, the web content is sent to the running process straight away. If it fails, it is sent to the recovery component.

N	*Hash Value (Signature)*	*Requested Resource*
1	D35C729762B3FC8795FB90631BCF64DEA061E33B	/cyberventues/st_proj.html
2	8AEEED714BACD7AECB74A054A5BE54A185CC9C52	/Hachett_trees07/trees.htm
3	D90FB8C0A13CEA7C87CD515E2277E299195A1436	/Hachett_trees07/amandahtml/kiana.htm
4	1E7707F952FCBF9627076FAE387C1E6685FA6192	/Hachett_trees07/amandahtml/natalie.htm
5	32CE36ED9039D3C2951350CFC5843BC145998CDA	/Hachett_trees07/amandahtml/nf/nat_1.GIF
...		
2941	56236652C0E32364638C5294B501E786BD0F4B91	/StartHere.jsp
2942	0709306BA800150FB58C6520F3310AEBD759AA16	/Store.jsp
...		
2961	8C967D27FB403E39848F46563070046EDA501529	/travel-styles.css
2962	DF9F21A89CB51BDFCADAFD6A0DE728AA2E152808	/tree2/backblue.gif
2963	14D5F457E32810A1D95D21FFE398AB39D110D177	/tree2/fade.gif
2964	B532E95DA8926D183002CA56BF26245EF49BD7E2	/rodman_guerriero/beginning.html
...		

Table 6.1: A partial list of original hashing measurement.

1	9D354FF4B08DDB0FE8BD0656692AE895C6F36887	/cyberventues/st_proj.html
2	83AC2737D5DBAEBA408E3513AF402158ACE0A4BE7	/Hachett_trees07/trees.htm
3	45A9CF81C355F5383E9CD18C3F7BDDFDB1062003	/Hachett_trees07/amandahtml/kiana.htm
4	0887CBC41B81A9C418EF1ED91C5AC4D4FA93D14A	/Hachett_trees07/amandahtml/natalie.htm
5	F688117B8752340851B89ECA5ECC853915815FB8	/Hachett_trees07/amandahtml/nf/nat_1.GIF
...		

Table 6.2: A partial list of hashing measurement after running malicious web content manipulation software.

Table 6.3 illustrates that a partial list of alerted web content has been compromised by running malicious web content manipulation software. For example, the requested page "/cyberventues/st_proj.html" had been altered by changing the following:

1. The value of HTML Title element "<title>Student Web Projects</title>" was altered to "<title>This is a different name</title>".

2. The colour of letters in the the linked image "stweb.gif" in Img element was altered. K, R, and J analysed the HTML source code of "/cyberventues/st_proj.html" by extracting the name of image element in this tag "<img src='images/stweb.gif' width=359 height=50>" and then altered the contents of this image object.

3. The white background of "/cyberventues/st_proj.html" was altered to yellow.

The original hash values of "/cyberventues/st_proj.html" are shown in Table 6.1. The altered hash values after running tampering are shown in Table 6.2. As can seen from Table 6.3 all alerted web content had been detected and recovered using the proposed WCVR system. It is suggested that the WCVR system has the capability to detect and recover web content that has been tampered with, and hence, the WCVR system satisfies the security objective as defined above.

Readers should note that Appendix A contains the following table:

1. Table 1: The whole list of alerted web content that are compromised by running malicious web content manipulation software.

N	HTTP Request	Altered Data (Actions)	Detection	Recovery
1	/cyberventues/st_proj.html	o Title changed to 'This is a different name'. o stweb.gif altered in paint, colour of letters changed. o Changed Background to Yellow.	YES	YES
2	/cyberventues/Hachett_trees07/trees.htm	o Amanda.html/Kiana.htm was deleted. o Amandahtml/vanessa.htm was deleted. o Amandahtml/micheal.htm was deleted.	YES	YES
3	/cyberventues/Hachett_trees07/amandahtml/kiana.htm	o Deleted everything between Html tags.	YES	YES
4	/cyberventues/rodpen07/penguins_rodman07.htm	o penline.jpg swapped around with hobopenguin.jpg o Underlined and altered colour of Miss Rodman's Third Graders	YES	YES
5	/cyberventues/rodman_about_both/rodman_about_both.htm	o Deleted CSS from this file.	YES	YES
6	/cyberventues/dummett_penguins06/main.htm	o Deleted the entire table.	YES	YES
7	/cyberventues/teeth_riddles_du/contents.htm	o Replaced the multiple images titled tooth5.gif, teethnn.gif and tttt.gif with a comment displaying the following characters, 'There is no picture'.	YES	YES
8	/cyberventues/give_mrs_early/html_give/olivia.htm	o Editing the four images sky colour to black. o Changing overall background colour to pink.	YES	YES
9	/cyberventues/early_tree06/tree.htm	o Deleted Hyperlinks and altered images so they have a big black line going through the middle	YES	YES
10	/cyberventues/pollution_movies05/main.htm	o Changed the following link 'http://www.apple.com/quicktime/products/qt/' to ' www.northumbria.ac.uk'.	YES	YES
11	/cyberventues/Hachett_trees07/amandahtml/natalie.htm	o Changing centre image to a red circle.	YES	YES

Table 6.3: A partial list of alerted web content that are compromised by running malicious web content manipulation software.

The following example will be used to explain how the integrity verification and recovery processes work. When K, R, and J students requested the `"Cart.jsp?itemID=2&count=17"` (see Figure 6.1 - d), the HTTP interface mechanism intercepted the HTTP request and analysed details of the request header (such as request name and URL parameters), as shown in Figure 6.3. After that, the mechanism extracted the request path to compare the hash value of original request (which is calculated by web register mechanism) with the one on a web server. In Figure 6.3, the log message `"Static Web Content (Cart.jsp) Integrity at the Server-Side has been successfully verified"` indicates that the integrity verification was successful. The HTTP interface mechanism generated log files, which includes a sequence of messages such as details of re-

quest header, the verification message (either for static or dynamic web content), hash value of response output, the state of online transaction database and recovery messages if the web content is tampered with.

In the next stage of verification, the http interface mechanism also intercepted, and analysed the output HTTP response of "Cart.jsp?itemID=2&count=17", as shown in Figure 6.3. The HTTP interface mechanism analysed the output HTTP response and extracted the following information: the state of connection, content type, content length, date, type of web server and response body. After that, this mechanism compared the original hash value of the generated dynamic web content (which was calculated by response hashing mechanism) with the hash value of the output HTTP response. In Figure 6.3, the log message "Dynamic Web Content Cart.jsp?itemID=2&count=17 Integrity at the Server-Side has been successfully verified" indicates that the integrity verification was successful.

In this context, the last log message ("Row is deleted" in Figure 6.3) denotes that the generated dynamic web content had been served to a target client. The details about this request are deleted from the DBMS online-transaction table.

K, R, and J focused on the generated dynamic web content from the JSP shopping cart application. For example, the Store.jsp web page (see Figure 6.1-presentation (b))includes the list of seven books. K, R, and J selected Moby Dick book (price: $5.80) by requesting URL presents in the following: "Cart.jsp?itemID=0&count=15" (Figure 6.1-presentation (c)). This kind of web content is dynamic because the HTTP request includes those URL parameters. K, R, and J then requested "Cart.jsp?itemID=2&count=17" (Figure 6.1-presentation (d)) and also requested "Cart.jsp?itemID=6&count=19" (Figure 6.1-presentation (e)).

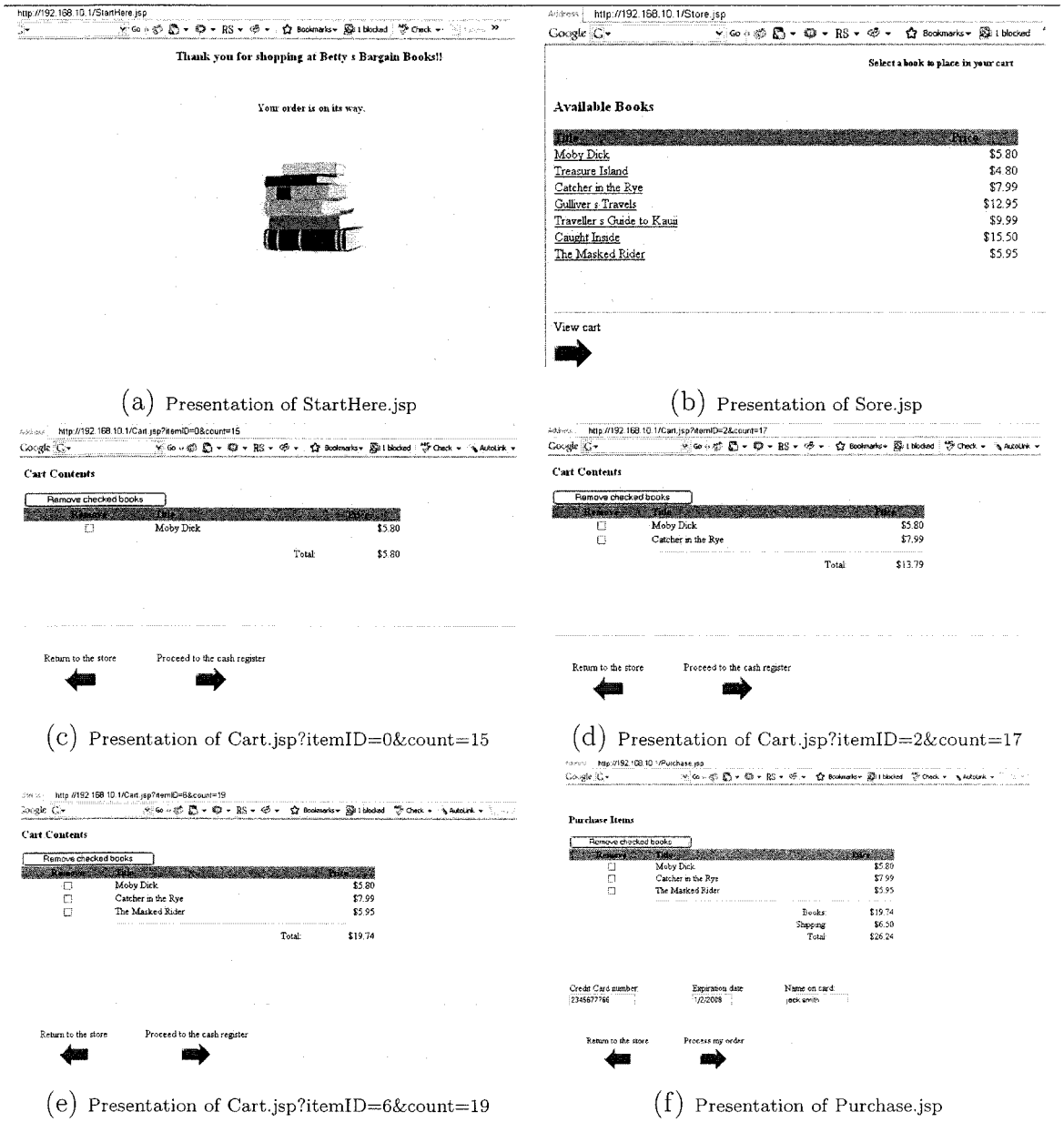
It can be seen from Figure 6.1 that K, R, and J generated three different dynamic web contents from the same source page "Cart.jsp" and that each generated HTTP response has a different hash value, as shown in the section "Response value of hash" in Figures 6.2, 6.3, and 6.4.

6.3.1.1 Section Conclusions

We have carried out the following two experiments to test the security objective (***How does WCVR system provide tamper detection and recovery in the server-side static and dynamic web content on Apache Tomcat and Microsoft IIS web servers***) of the proposed WCVR system:

- Experiment 1: To investigate the tamper detection and recovery in a server-side static and dynamic web content on Apache Tomcat web server.
- Experiment 2: To investigate the tamper detection and recovery in a server-side static web content on Microsoft IIS web server.

Three computing students at the Computer Forensics lab at Northumbira University picked over five hundred web requests from the (i) UK Hillside Primary school web site and (ii) Borland JBuilder JSP shopping cart. The students identified a potential victim list of target web resources and manually confirmed exploitable flaws in the identified web resources. Over 70 tampering attacks were launched against the designated directories of the suggested web sites hosted on Tomcat and IIS web servers. The results of this experimental study have shown that the proposed system (WCVR) may provide a high coverage of detection and recovery and a low level of overhead times.



(a) Presentation of StartHere.jsp

(b) Presentation of Sore.jsp

(c) Presentation of Cart.jsp?itemID=0&count=15

(d) Presentation of Cart.jsp?itemID=2&count=17

(e) Presentation of Cart.jsp?itemID=6&count=19

(f) Presentation of Purchase.jsp

Figure 6.1: A sequence of static and dynamic web pages from JSP shopping cart application have been requested.

Request Details

Request: GET /Cart.jsp?itemID=0&count=15 HTTP/1.1

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, application/x-shockwave-flash, */*

Referer: http://phdmachine/Store.jsp

Accept-Language: en-us

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)

Host: phdmachine

Connection: Keep-Alive

Cookie: JSESSIONID=326ADA5C45E7B5D4C752E7597E83FE52DB

Static Web Content (Cart.jsp) Integrity at the Server-Side has been successfully verified

Response Details

HTTP/1.1 200 OK

Content-Type: text/html

Content-Length: 1345

Date: Tue, 18 Dec 2007 19:09:01 GMT

Server: Apache-Coyote/1.1

<HTML><!--ShoppingCartappCart.jsp--
><BODYBACKGROUND="images/greenpaper.gif"><FORMMETHOD=POST><H3>CartContents</
H3><INPUTTYPE=HIDDENNAME="pageCount"VALUE="15"><INPUTTYPE=submitNAME="Re
move"VALUE="Removecheckedbooks"><TABLEWIDTH="60%"CELLSPACING="0"CELLPADDI
NG="1"><THALIGN="CENTER"BGCOLOR="#808080">Remove</TH><THALIGN="LEFT"BGC
OLOR="#808080">Title</TH><THBGCOLOR="#808080">Price</TH><TR><TDALIGN="CENTE
R"><INPUTTYPE=CHECKBOXNAME=REMOVE_BOOK0></TD><TD>**MobyDick**</TD><TDAL
IGN="RIGHT">**$5.80**</TD></TR><TR><TDALIGN="CENTER"><INPUTTYPE=CHECKBOXNA
ME=REMOVE_BOOK1></TD></TABLE>

</FORM>

<HR><TABLEC
ELLPADDING=30><TR><TDALIGN="CENTER">Returntothestore
<AHREF="Store.jsp"><I
MGSRC="images/larrow.gif"ALT="goback"BORDER=0></A></TD><TDALIGN="CENTER">Proc
eedtothecashregister
<AHREF="Purchase.jsp"><IMGSRC="images/rarrow.gif"ALT="proceed"
BORDER=0></A></TD></TR></TABLE></BODY></HTML>

Response value of hash value:832C12713636C6545666C42E8DDD50283562EABBDE2

Dynamic Web Content(Cart.jsp?itemID=0&count=15) Integrity at the Server-Side has been successfully verified

Row is deleted

Figure 6.2: Log file: `Cart.jsp?itemID=0&count=15`

Request Details

Request: GET /Cart.jsp?itemID=2&count=17 HTTP/1.1

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, application/x-shockwave-flash, */*

Referer: http://phdmachine/Store.jsp

Accept-Language: en-us

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)

Host: phdmachine

Connection: Keep-Alive

Cookie: JSESSIONID=2AA545E7B5D4C752797E3FE55E02F36C

Static Web Content (Cart.jsp) Integrity at the Server-Side has been successfully verified

Response Details

HTTP/1.1 200 OK

Content-Type: text/html

Content-Length: 1345

Date: Tue, 18 Dec 2007 19:09:21 GMT

Server: Apache-Coyote/1.1

<HTML><!--ShoppingCartappCart.jsp--
><BODYBACKGROUND="images/greenpaper.gif"><FORMMETHOD=POST><H3>CartContents</
H3><INPUTTYPE=HIDDENNAME="pageCount"VALUE="17"><INPUTTYPE=submitNAME="Re
move"VALUE="Removecheckedbooks"><TABLEWIDTH="60%"CELLSPACING="0"CELLPADDI
NG="1"><THALIGN="CENTER"BGCOLOR="#808080">Remove</TH><THALIGN="LEFT"BGC
OLOR="#808080">Title</TH><THBGCOLOR="#808080">Price</TH><TR><TDALIGN="CENTE
R"><INPUTTYPE=CHECKBOXNAME=REMOVE_BOOK0></TD><TD>**MobyDick**</TD><TDAL
IGN="RIGHT">**$5.80**</TD></TR><TR><TDALIGN="CENTER"><INPUTTYPE=CHECKBOXNA
ME=REMOVE_BOOK1></TD><TD>**CatcherintheRye**</TD><TDALIGN="RIGHT">**$7.99**</TD><
/TR><TR><TD></TD><TD><HR></TD><TD><HR></TD></TR><TR><TD></TD><TDALIGN="
RIGHT">**Total:**</TD><TDALIGN="RIGHT">**$13.79**</TD></TABLE>

</FORM>
<
BR>
<HR><TABLECELLPADDING=30><TR><TDALIGN="CENTER">Returntothestore
<AHREF="Store.jsp"><IMGSRC="images/larrow.gif"ALT="goback"BORDER=0></A></TD><TD
ALIGN="CENTER">Proceedtothecashregister
<AHREF="Purchase.jsp"><IMGSRC="images/ra
rrow.gif"ALT="proceed"BORDER=0></A></TD></TR></TABLE></BODY></HTML>

Response value of hash value:903E12713636C8646C42F9E6150283562E48ACF8

Dynamic Web Content(Cart.jsp?itemID=2&count=17) Integrity at the Server-Side has been successfully verified

Row is deleted

Figure 6.3: Log file: `Cart.jsp?itemID=2&count=17`

Request Details

Request: GET /Cart.jsp?itemID=6&count=19 HTTP/1.1

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, application/x-shockwave-flash, */*

Referer: http://phdmachine/Store.jsp

Accept-Language: en-us

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)

Host: phdmachine

Connection: Keep-Alive

Cookie: JSESSIONID=2AA545E7B5D4C752797E3FE55E02F36C

Static Web Content (Cart.jsp) Integrity at the Server-Side has been successfully verified

Response Details

HTTP/1.1 200 OK

Content-Type: text/html

Content-Length: 1473

Date: Tue, 18 Dec 2007 19:09:24 GMT

Server: Apache-Coyote/1.1

Response value:<HTML><!--ShoppingCartappCart.jsp--><BODYBACKGROUND="images/greenpaper.gif"><FORMMETHOD=POST><H3>CartContents</H3><INPUTTYPE=HIDDENNAME="pageCount"VALUE="19"><INPUTTYPE=submitNAME="Remove"VALUE="Removecheckedbooks"><TABLEWIDTH="60%"CELLSPACING="0"CELLPADDING="1"><THALIGN="CENTER"BGCOLOR="#808080">Remove</TH><THALIGN="LEFT"BGCOLOR="#808080">Title</TH><THBGCOLOR="#808080">Price</TH><TR><TDALIGN="CENTER"><INPUTTYPE=CHECKBOXNAME=REMOVE_BOOK0></TD><TD>**MobyDick**</TD><TDALIGN="RIGHT">**$5.80**</TD></TR><TR><TDALIGN="CENTER"><INPUTTYPE=CHECKBOXNAME=REMOVE_BOOK1></TD><TD>**CatcherintheRye**</TD><TDALIGN="RIGHT">**$7.99**</TD></TR><TR><TDALIGN="CENTER"><INPUTTYPE=CHECKBOXNAME=REMOVE_BOOK2></TD><TD>**TheMaskedRider**</TD><TDALIGN="RIGHT">**$5.95**</TD></TR><TR><TD></TD><TD><HR></TD><TD><HR></TD></TR><TR><TD></TD><TDALIGN="RIGHT">**Total:**</TD><TDALIGN="RIGHT">**$19.74**</TD></TABLE>

</FORM>

<HR><TABLECELLPADDING=30><TR><TDALIGN="CENTER">Returntothestore
<AHREF="Store.jsp"><IMGSRC="images/larrow.gif"ALT="goback"BORDER=0></A></TD><TDALIGN="CENTER">Proceedtothecashregister
<AHREF="Purchase.jsp"><IMGSRC="images/rarrow.gif"ALT="proceed"BORDER=0></A></TD></TR></TABLE></BODY></HTML>

Response value of hash value:4922DD4563B0E377356163FB3026E4F4AC414E02

Dynamic Web Content(Cart.jsp?itemID=6&count=19) Integrity at the Server-Side has been successfully verified

Row is deleted

Figure 6.4: Log file: `Cart.jsp?itemID=6&count=19`

6.3.2 Case Study for Micro-benchmarking Performance

We measured the runtime performance of the web register mechanism with a set of micro-benchmarks. We measured the latencies of web register mechanism in two different cases, namely, SHA-1 (10 digits) and SHA1-extended (16 digits). In the SHA-1 case, we calculated the hash value of a web content using SHA-1 function (10 digits). The SHA1-extended represented the case when we calculated the hash value of a web content by SHA-1 function where number of digits was 16. Since the goal is to measure the latency, we ran the web register mechanism 15 times over 200 entries of different sizes for every case (SHA-1 and SHA1-extended) using MS Windows XP Professional. The implementation of the micro-benchmarks is based on the HBench framework industrial standard (132).

An illustration of results is presented in Table 6.4. It should be noted from this table that the web register overhead in the case of SHA-1 (10 digits) is low – the average running time was 1.4274 seconds (representing the average of time taken to run 15 trails), which is less time when using SHA1-extended (2.2176 seconds). These cases do not only measure the overhead of the hash value itself, it also measured all functions in a web register mechanism for both cases (SHA-1 and SHA1-extended).

We have concluded that the SHA1-extended is the most costly in performance terms. This is understandable, because the SHA1-extended contains 16 digits instead of 10 digits. Readers should note that this work as detailed in this thesis is more concerned with security than performance.

It is anticipated that performance gains can be expected from an industrial standard web server. In the case of static content, the web register is able to hash the web content before a request is responded to, however in the case of dynamic content, the hashing is required at the time of delivery and hence requires more computerised effort.

We have also presented the registry performance of a web content as a function of file sizes. We measured the web register mechanism running time for both: SHA-1 and SHA1-extended, varying the input file sizes. The results are shown

in Table 6.5. When the file size is large, the difference in the hashing overhead can be significant. For example, a 64 Kilobytes file when using SHA1-extended takes about 12.47 milliseconds, where it takes about 3.2 milliseconds for SHA-1. Furthermore, when using SHA1-extended, a 13 Megabytes file records 1531 milliseconds performance overhead, but when using SHA-1, the same file records 620.067 milliseconds performance overhead.

In Figure 6.5, the performance overhead (ms) with the SHA1-extended case is represented by an unbroken curve, while the performance overhead (ms) with the SHA1 case is represented by a dashed line. The horizontal axis represents the file sizes in byte, and the left vertical axis represents the overhead running time in milliseconds. As expected, the performance overhead has a direct correspondence to the file size, i.e. the larger file size is the greater performance overhead.

Table 6.4: Overhead of a web register mechanism

Web Register Call	***Overhead (ms)***
SHA1-extended (16 digits)	2217.6 (2.2176s)
SHA-1 (10 digits)	1427.4 (1.4274s)

Table 6.5: Registry Performance for both SHA1-extended and SHA-1 as compared with file sizes.

File Size (Byte)	***Overhead (ms) with SHA1-extended***	***Overhead (ms) with SHA-1***
1KB	0.64	0.627
16KB	5.13	2.13
64KB	12.47	3.2
2MB	163.73	118.73
5MB	348	251.97
13MB	1531	620.067

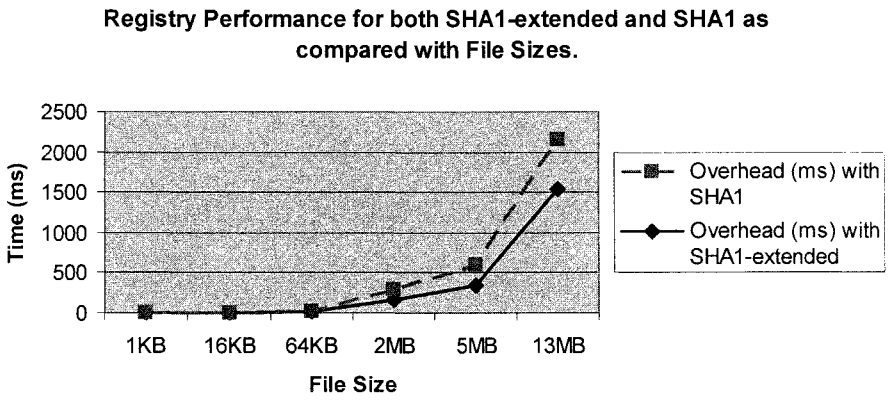


Figure 6.5: A linear chart of registry Performance for both SHA1-extended and SHA-1 as compared with file sizes.

As shown in Figure 6.5, longer keys take much more computing resource to decrypt, and hence make them less vulnerable to attack. However, the SHA1-extended is also more costly in performance terms, but this is the cost that legitimate users pay for higher levels of security. The impact of hashing and encryption are issues that increase the overhead and they are rarely considered in the area of web engineering and design. When using the encryption and hashing, some users have observed that the CPU overhead of sending and receiving encrypted requests, and the hashing verification to be as high as 100 to 200 milliseconds per request, easily overwhelming any other processing (48). This overhead varies widely by implementation, key length and other factors, but it is always costly in performance terms. In this experimental study, we measured the performance for two cases: SHA-1 (10 digits), and SHA1-extended (16 digits).

It should also be noted that Appendix B contains the following two tables:

1. Table 1: List of all entries that were measured using SHA-1 function.
2. Table 2: List of all entries that had been measured using SHA1-extended function.

6.3.3 End-to-End Performance Evaluation

The testing environment is composed of two web servers: Apache 1.3.29 with Tomcat container 5.01 on MS Windows Server 2003 and IIS 6.0 on MS Windows Server 2003. The two web servers contain a copy of target web site and shopping cart application. Over 70 attacks were performed against the server-side generated static and dynamic web content security properties. These attacks exploited different types of vulnerabilities that allowed for the modification of files in the designated directories of a web server. As a result, all the known attacks launched against the web content were detected and recovered by the WCVR system.

A load test can be used to test an application's robustness and performance, as well as its hardware and bandwidth capacities. In these experiments, we used the Neoload application which is a stress and load testing tool to (i) test a web site's vulnerability to crashing under load and (ii) check response times under the predicted load.

As the verification and recovery processes are performed online in real-time, it should induce a time overhead in the service. The results presented in this section have been obtained on the same set of requests, using the same network architecture.

Three undergraduate computing students individually conducted three experimental studies (including case study K, case study R, and case study J) to measure the end-to-end performance with the (i) proposed WCVR system, (ii) DSSA system, and (iii) without any verification system.

The duration of the test was dedicated by the requirements of the Neoload testing. The run-time policy was ramped-up (i.e. generating a number of virtual users that increases throughout the test until it reaches the specified maximum) from 2 users (initial user number at the same time) adding 2 users every 2 minutes. This is useful for checking the server's behaviour under an increasing load. The virtual users were connecting at 100Mbps through a local network.

All these measurements were performed from the client point of view (i.e. all durations show the time between a request and the reception of its answer). Each

row in the below tables displays the average response time (request), maximum response time (request), and minimum response time (request) in seconds, of all requests during the test and average page response time for all pages where each page may contain a number of requests. The average response time is the mean-time necessary to process a request by each web server when the proxy, browser, and the WCVR system are active. The activation of the WCVR implies that HTTP request-response communication has changed. The activation of the of the recovery component implies that the server-side static and/or dynamic web content has been tampered with. The communications (network response time) are parts of the measured times.

6.3.3.1 Case Study K

This study consisted of two parts: static web content, and dynamic web content. We have summarised the results in Table 6.6 and Table 6.7, respectively.

Case Study K- Static Web Content

Experiment 3 and 4 were carried out using MS Windows environment (i.e. MS Windows XP Professional for clients machines, and MS Windows Server 2003 for a server) with IIS and Tomcat web servers. Experiment 3 is designed to show end-to-end performance (i.e. verification of the integrity of server-side static web content; the recovery response if the web content has been tampered with; the network speed; the web server response; and the web browser response) via (i) the proposed WCVR system and (ii) the existing DSSA system on IIS web server. Experiment 4 is designed to show end-to-end performance (i.e. verification of the integrity of server-side static web content; the recovery response if the web content has been tampered with; the network speed; the web server response; and the web browser response) via (i) the proposed WCVR system and (ii) the existing DSSA system on Tomcat web server.

In experiment 3, 54645 requests were created, 2575 web pages were served and 572.29MB (total throughput) were received by the client. The number of virtual users launched was between 108 and 221. In experiment 4, 20626 requests were created, 10482 web pages were served, and 211.73MB (total throughput) were

received by the client. The number of virtual users launched was between 30 and 93.

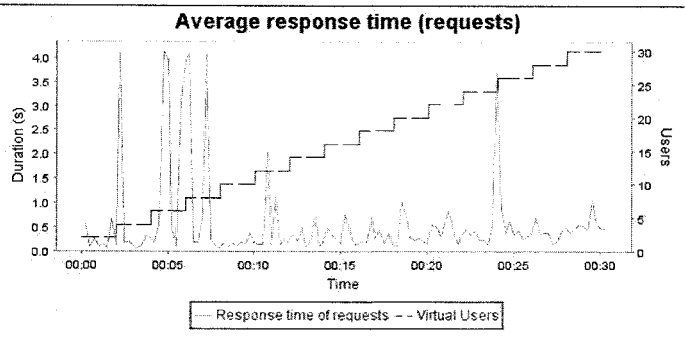
System	Web Server	Graph Min (request)	Average Response Time (request)	Graph Max (request)	Average 90% (request)	Average Page Response Time
DSSA	IIS	0.076	0.357	4.14	0.318	0.309
WCVR	IIS	0.084	0.307	4.39	0.292	0.607
DSSA	Tomcat	0.106	0.834	15.4	0.669	1.63
WCVR	Tomcat	0.046	0.66	60.7	0.661	1.88

Table 6.6: Case Study K- Static Web Content: Comparison between the response times through DSSA and WCVR systems, in seconds, of all requests during the test on IIS and Tomcat web servers.

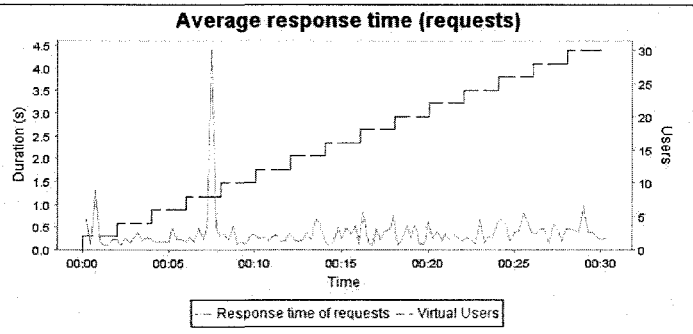
A comparison of the WCVR and DSSA systems indicates that the end-to-end performance of the WCVR (on IIS and Tomcat web servers) is better than the end-to-end performance of the DSSA. As illustrated in Table 6.6, the average response time (request) through WCVR was 0.307 seconds on IIS and was 0.66 seconds on Tomcat, whereas the average response time (request) through DSSA was 0.357 seconds on IIS and was 0.833 seconds on Tomcat.

In Figure 6.6, the average response times (request) are represented by an unbroken curve, while the number of virtual users is represented by a dashed line. The horizontal axis represents the duration of test in minutes, the left vertical axis represents the running time in seconds and the right vertical axis shows the number of virtual users. In the four graphes the number of virtual users increases closing to a linear state as the duration of test increases as the test adds 2 users every 2 minutes. These graph indicate how the number of users changes with time during the test.

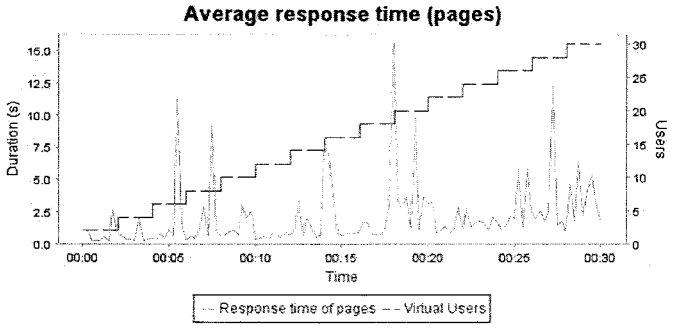
As shown in Figure 6.6 : graph (a-DSSA), the maximum response through the DSSA system (on IIS web server) was at 1 minute, 5 minutes, 6.5 minutes, and 8 minutes (with a maximum response time of 4.14 seconds). Whereas in Figure 6.6 : graph (b-WCVR), the maximum response through the WCVR system (on IIS web server) was only at 8 minutes (with a maximum response time of 4.39 seconds).



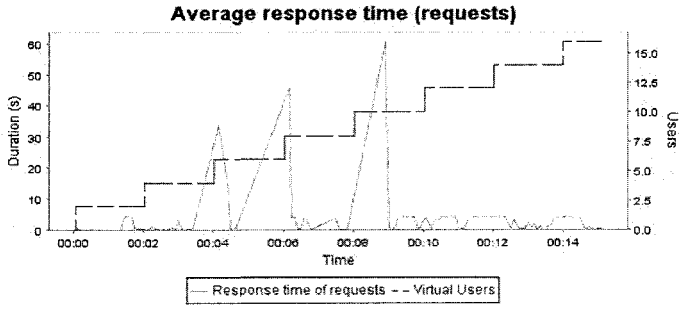
(a) Average response time through DSSA on IIS



(b) Average response time through WCVR on IIS



(c) Average response time through DSSA on Tomcat



(d) Average response time through WCVR on Tomcat

Figure 6.6: Case Study K - Static Web Content: Graphs for the response times (request) through the DSSA and WCVR systems on IIS and Tomcat web servers, in seconds, of all requests during the test

The graph (c-DSSA) in Figure 6.6 shows the minimum response time was 0.106 seconds at the first minute; where there was a sharp increase of response times to reach the maximum value (15.4 seconds) at the 18th minute. The standard deviation was low (1.09 seconds) indicating that the response times showed little variation. Whereas, the graph (d-WCVR) in Figure 6.6 shows the minimum response time is 0.046 seconds at the first minute; where there is a sharp increase of response times to reach the maximum value (60.7 seconds) at the 9th minute because:

- There was insufficient memory on the server, which caused a disconnection of the communication between the server and client.

After that, the response time decreases dramatically closing to a constant state. The standard deviation was high (3.72 seconds) indicating that the response times varied widely (see graph (d-WCVR) in Figure 6.6).

It would appear that the WCVR is less costly in performance terms when verifying the server-side static web content against tampering attacks on IIS and Tomcat web servers.

Case Study K- Dynamic Web Content

Experiment 5 was carried out on MS Windows environment (i.e. MS Windows XP Professional for clients machines and MS Windows Server 2003) using a Tomcat web servers. This experiment is designed to show end-to-end performance (i.e. verification of the integrity of server-side static web content; the recovery response if the web content has been tampered with; the network speed; the web server response; and the web browser response) via (i) the proposed WCVR system and (ii) without any verification system on Tomcat web server. In this study, the response times for Scenario A (without any verification system) and Scenario B (with the WCVR system) were collected.

Readers should note that the testing of dynamic web applications relies on HTTP requests that can include parameters which are specific for each user. The values of such parameters can be different for different users of the same type and

can change throughout the session. For example, a web server can send a session variable in response to the first request from a user. This variable is inserted to the subsequent requests and is used to identify that user.

In this case study, 16396 hits were created, 14136 web pages were served, and 29.29MB (total throughput) were received by the client. Number of virtual users launched was between 150 and 474.

System	Web Server	Graph Min (request)	Average Response Time (request)	Graph Max (request)	Average Page Response Time
Without any verification system	Tomcat	0.032	0.527	4.08	0.634
WCVR	Tomcat	0.011	0.437	5.29	0.505

Table 6.7: Case Study K - Dynamic Web Content: Comparison between the response times in seconds, of all requests during the test on Tomcat web server.

Table 6.7 illustrates that the average response time (request) through WCVR was 0.437 seconds on Tomcat. In Scenario A (without any verification system), the average response time (request) was 0.527 seconds on Tomcat. Therefore, the WCVR satisfies the performance objective for verification of integrity of dynamic web content.

The results indicate that the average response time when using the WCVR system is better than when not using any verification system - it is suggested that the reason for this is as follows:

- The HTTP connection problem: For example, at the first 3 minutes during the test (see graph (a) in Figure 6.7), student K requested the wrong URL (`GET http://192.168.10.1/StartHere.jsp HTTP/1.1`) indicating the communication between the client and the server was disconnected. Consequently this led to an increase in the average response time. However, the correct URL was (`GET http://192.168.10.1:8081/StartHere.jsp HTTP/1.1`). Note, K did not add the port number (`8081`) of connection in the URL request.

The graph (a) in Figure 6.7 represents the curve of response times throughout this test without any verification system on Tomcat. As shown in graph (a), the minimum response time was 0.032 seconds and, the variation of response times was relatively low.

The graph (b) in Figure 6.7 shows the minimum response time through WCVR was 0.011 seconds on Tomcat. In addition, the response times did not vary widely.

(a) Average response time through without verification system on Tomcat (b) Average response time through WCVR system on Tomcat

Figure 6.7: Case Study K- Dynamic Web Content: Graphs for response time curves through without verification system and the WCVR on Tomcat, in seconds, of all requests during the test.

6.3.3.2 Case Study R

This study consisted of two parts: static web content, and dynamic web content. We have summarised the results in Table 6.8 and Table 6.9, respectively.

Case Study R- Static web content

Experiment 3 and 4 were carried out on MS Windows environment (i.e. MS Windows XP Professional for clients machines and MS Windows Server 2003) using IIS and Tomcat web servers. In experiment 3, 9130 hits were created, 6096 web pages were served, and 57.22MB (total throughput) were received by the client. The number of virtual users launched was between 36 and 44. In experiment 4, 9023 hits were created, 5882 web pages were served, and 34.75MB (total throughput) were received by the client. The number of virtual users launched was over 45.

System	Web Server	Graph Min (request)	Average Response Time (request)	Graph Max (request)	Average 90% (request)	Average Page Response Time
DSSA	IIS	0.085	2.77	48	2.46	3.56
WCVR	IIS	0.086	1.56	5.98	1.49	2.33
DSSA	Tomcat	0.097	1.9	5.57	1.86	2.9
WCVR	Tomcat	0.101	3.78	5.44	3.94	3.94

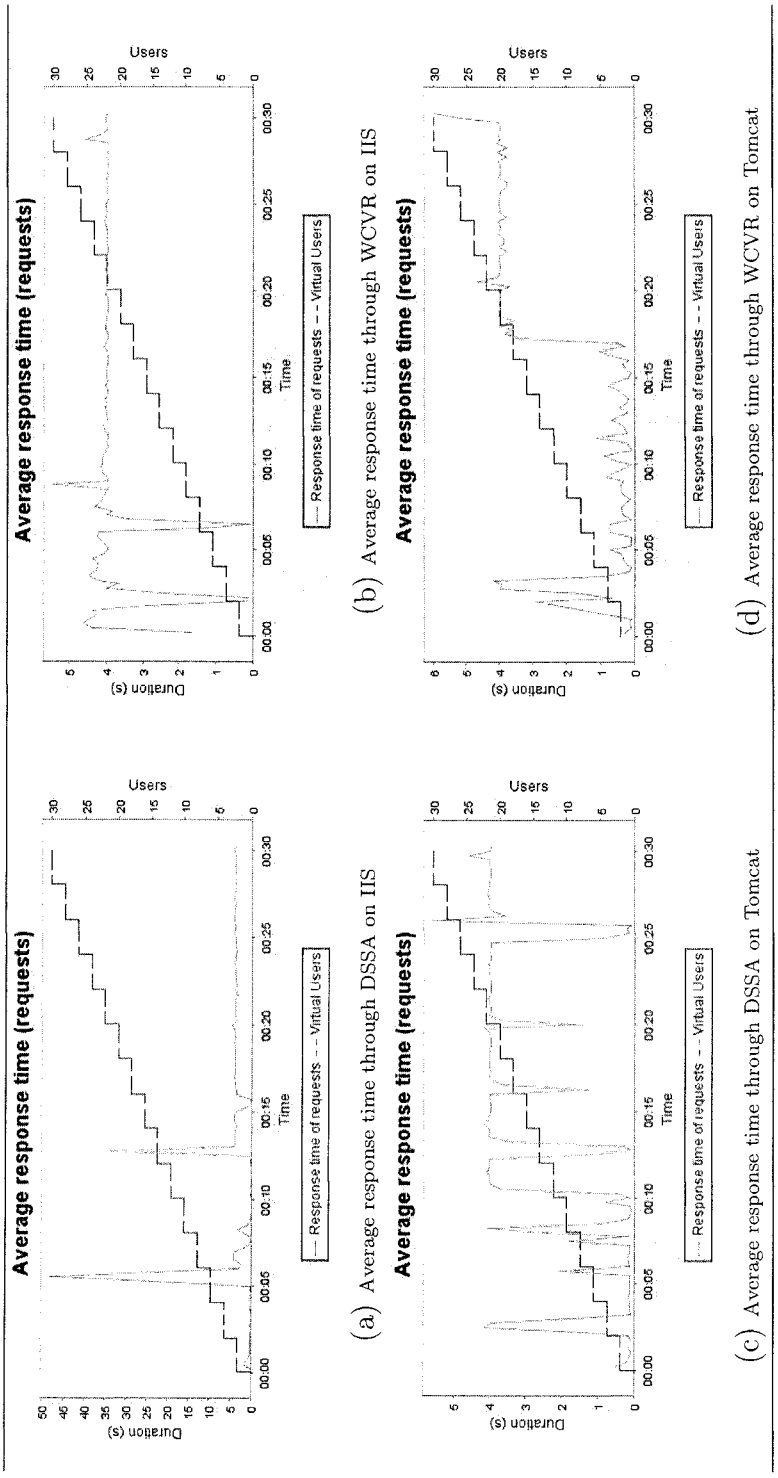
Table 6.8: Case Study R - Static web content: Comparison between the response times through DSSA or WCVR systems, in seconds, of all requests during the test on IIS and Tomcat web servers.

Table 6.8 demonstrates that the average response time (request) of the WCVR on IIS web server was 1.56 seconds and on Tomcat was 3.78 seconds. Whereas, the average response time (request) of the DSSA on IIS was 2.77 seconds and on Tomcat was 1.90 seconds. Therefore, the average response time of the WCVR system was less than the overhead times of the DSSA on IIS web server. However, the average response time (request) of the WCVR on Tomcat was higher than a DSSA. The total errors that occurred in the HTTP conversation via the WCVR on Tomcat during 30 minutes was much higher than the total errors that occurred in the DSSA on Tomcat.

If number of errors increases, then the average response time increases and hence the overhead time increases. The results indicate that the end-to-end performance when using the DSSA system on Tomcat is better than when using the WCVR system - it is suggested that this occurs because the following reasons:

1. Miscellaneous I/O error when connecting to the server.
2. Error when connecting to the server for insufficient memory. Indicates a server-side error occurring when attempting to bind a socket.

The graph (c - DSSA) in Figure 6.8 illustrates the maximum response time (request) through the DSSA system on Tomcat web server was 5.57 seconds at the 26th minute. There were small variations in the response times throughout this test. Whereas, the graph (d - WCVR) in Figure 6.8 shows the variations of response times through the WCVR system (on Tomcat web server) were highly variable from the first minute to the 9th minute closing to a constant state from the 9th minute to the 30th minute.



(a) Average response time through DSSA on IIS

(b) Average response time through WCVR on IIS

(c) Average response time through DSSA on Tomcat

(d) Average response time through WCVR on Tomcat

Figure 6.8: Case Study R - Static web content: Graphs for response time (request) curves through the DSSA and the WCVR on IIS and Tomcat servers, in seconds, of all requests during the test.

Case Study R- Dynamic Web Content

Experiment 5 was carried out on MS Windows environment using a Tomcat web server. This experiment is designed to show end-to-end performance (i.e. verification of the integrity of server-side static web content; the recovery response if the web content has been tampered with; the network speed; the web server response; the web browser response) via (i) the proposed WCVR system and (ii) without any verification system on Tomcat web server. In this study, the response times for Scenario A (with the WCVR system) and Scenario B (without any verification system) were collected.

In case study, 2941 hits were created, 2832 web pages were served, and 1.81MB (total throughput) were received by the client. The number of virtual users launched was between 107 and 158.

System	Web Server	Graph Min (request)	Average Response Time (request)	Graph Max (request)	Average Page Response Time
Without any verification system	Tomcat	0.014	3.49	4.68	3.62
WCVR	Tomcat	0.118	11.7	208.6	12.1

Table 6.9: Case Study R- Dynamic Web Content: Graphs for response time curves through without verification system and the WCVR on Tomcat, in seconds, of all requests during the test.

The results indicate that the average response time when not using any verification system is better than when using the WCVR system. Table 6.9 illustrates that the average response time (request) through the WCVR was 11.7 seconds on Tomcat. In Scenario B (when not using any verification system), the average response time (request) was 3.49 seconds on Tomcat.

The graph (a) in Figure 6.9 shows the maximum response time when not using any verification system (on Tomcat web server) was 4.68 seconds at the 1st minute. The graph (b) in Figure 6.9 shows the the maximum response time when using the WCVR system was 208.6 seconds at the 8th minute, because the

WCVR system was disconnected, the actual test started from the 4th minute. This kind of error would tend to increase the average response time.

6.3.3.3 Case Study J

This study consisted of two parts: static web content and dynamic web content. We have summarised the results in Table 6.10 and Table 6.11, respectively.

Case Study J - Static web content

Experiment 3 and 4 were carried out using MS Windows environment (i.e. MS Windows XP Professional for clients machines and MS Windows Server 2003) with IIS and Tomcat web servers. Experiment 3 is designed to show end-to-end performance (i.e. verification of the integrity of server-side static web content; the recovery response if the web content has been tampered with; the network speed; the web server response; the web browser response) via (i) the proposed WCVR system and (ii) the existing DSSA system on IIS web server. Experiment 4 is designed to show end-to-end performance via (i) the proposed WCVR system and (ii) the existing DSSA system on Tomcat web server.

In experiment 3, 9130 hits were created, 6096 web pages were served and 57.22MB (total throughput) were received by the client. The number of virtual users launched was between 114 and 207. In experiment 4, 11041 hits were created, 6116 web pages were served, and 89.84MB (total throughput) were received by the client. The number of virtual users was between 88 and 89.

A comparison of the WCVR and DSSA systems indicates that the end-to-end performance of the WCVR (on both web servers: IIS, and Tomcat) is much better than that on the DSSA. As illustrated in Table 6.10, the average response time (request) through WCVR was 1.06 seconds on IIS and was 1.93 seconds on Tomcat, whereas the average response time (request) through DSSA was 4.03 seconds on IIS and was 3.86 seconds on Tomcat.

In Figure 6.10, the average response times (request) are represented by an unbroken curve, whilst the number of virtual users is represented by a dashed line. The horizontal axis represents the duration of test in minutes, the left

(a) Average response time through without verification mechanism on Tomcat (b) Average response time through WCVR on Tomcat

Figure 6.9: Case Study R- Dynamic Web Content: Graphs for response time curves through without verification system and the WCVR on Tomcat, in seconds, of all requests during the test.

System	Web Server	Graph Min (request)	Average Response Time (request)	Graph Max (request)	Average 90% (request)	Average Page Response Time
DSSA	IIS	0.165	4.03	4.48	4.06	3.84
WCVR	IIS	0.073	1.06	4.12	0.951	1.63
DSSA	Tomcat	0.117	3.86	4.71	4.04	3.96
WCVR	Tomcat	0.07	1.93	90.3	0.834	3.46

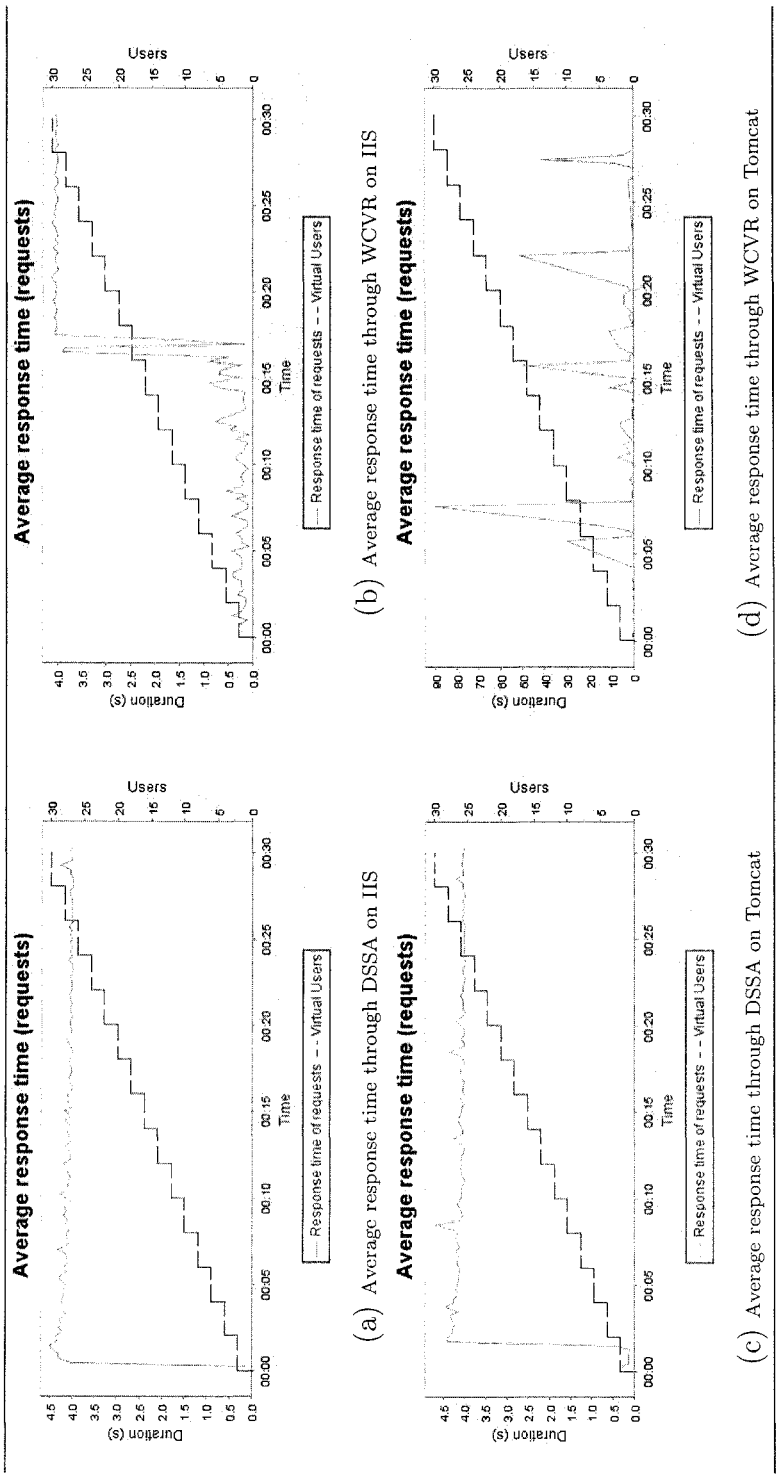
Table 6.10: Case Study J - Static web content: Comparison between the response times through DSSA or WCVR systems, in seconds, of all requests during the test on IIS and Tomcat web servers

vertical axis represents the running time in seconds and the right vertical axis shows the number of virtual users. In the four graphes, the number of virtual users increases closing to a linear state as the duration of test increases. The test adds 2 users every 2 minutes. These graph indicate how the number of users changes with time during the test.

The graph (a-DSSA) in Figure 6.10 illustrates that the maximum response through the DSSA system (on IIS web server) was at 90 seconds (with a maximum response time of 4.48 seconds). After the maximum response time at the 90th seconds, the response time closed to a constant state. The graph (b-WCVR) in Figure 6.10 illustrates the maximum response through the WCVR system (on IIS web server) was only at the 17th minute (with a maximum response time of 4.12 seconds).

The graph (c-DSSA) in Figure 6.10 shows the maximum response time was 4.71 seconds at the 2nd minute. The response times showed little variation. Whereas, the graph (d-WCVR) in Figure 6.10 illustrates the maximum response time was 90.3 seconds at the 8th minute. The end-to-end performance obtained is highly variable; factors such as insufficient memory and not requesting the correct URL can severely limit the performance through the WCVR system, and therefore, the response times showed much variation.

It would appear that the WCVR is less costly in performance terms when verifying the server-side static web content against tampering attacks on IIS and



(a) Average response time through DSSA on IIS

(b) Average response time through WCVR on IIS

(c) Average response time through DSSA on Tomcat

(d) Average response time through WCVR on Tomcat

Figure 6.10: Case Study J - Static web content: Graphs for response time (request) curves through DSSA and the WCVR on IIS and Tomcat servers, in seconds, of all requests during the test.

Tomcat web servers.

Case Study J - Dynamic Web Content

Experiment 5 was carried out on MS Windows environment (i.e. MS Windows XP Professional for clients machines and MS Windows Server 2003) using a Tomcat web server. This experiment is designed to show end-to-end performance via (i) the proposed WCVR system and (ii) without any verification system on Tomcat web server. In this study, the response times for Scenario A (without any verification system) and Scenario B (with the WCVR system) were collected.

In case study, 5421 hits were created, 5397 web pages were served and 1.55MB (total throughput) were received by client. the number of virtual users launched was between 198 and 204.

System	Web Server	Graph Min (request)	Average Response Time (request)	Graph Max (request)	Average Page Response Time
Without any verification system	Tomcat	0.029	3.14	4.86	3.24
WCVR	Tomcat	0.219	3.95	6.01	3.97

Table 6.11: Case Study J - Dynamic Web Content: Comparison between the response times through without verification system or WCVR systems, in seconds, of all requests during the test on Tomcat web server.

Table 6.11 illustrates that the average response time (request) through WCVR was 3.95 seconds on Tomcat. In Scenario A (without any verification system), the average response time (request) was 3.14 seconds on Tomcat. Therefore, the WCVR satisfies the performance objective for verification of integrity of dynamic web content.

The results indicate that the average response time when not using any verification system (Scenario A) is better than when using the WCVR system (Scenario B).

The graph (a) in Figure 6.9 represents the curve of response times without any verification system on Tomcat. As shown in graph (a), the maximum response

time was 4.86 seconds with the variation of response times was relatively low. The graph (b) in Figure 6.9 shows the maximum response time through WCVR was 6.01 seconds on Tomcat. The response times do not vary widely. As the WCVR system was disconnected, the actual test through the WCVR started from the 4th minute. This kind of error would tend to increase the average response time.

(a) Average response time through without verification mechanism on Tomcat (b) Average response time through WCVR on Tomcat

Figure 6.11: Case Study J - Dynamic Web Content: Graphs for response time curves through without verification system and the WCVR on Tomcat, in seconds, of all requests during the test.

6.4 Further Discussions and Conclusions

The experimental results indicate that not only is the WCVR able to ensure the survivability of dynamic web content, it is also able to ensure the survivability of static content at a faster and more reliable rate.

In the case of static web content, the results of experiential studies (K, R, and J) appear to indicate that the average end-to-end performance when using the WCVR system (on IIS and Tomcat web servers) is better than when using the DSSA system. As shown Figure 6.12 the average response time through the WCVR was 2.12 seconds on Tomcat and 0.977 seconds on IIS, whilst the average of response time through DSSA was 2.198 seconds on Tomcat and was 2.39 seconds on IIS. The WCVR results (through the WCVR) are lower than the response times through the DSSA on both web server applications.

In case of dynamic web content, the results of experiential studies (K, R, and J) appear to indicate that the average end-to-end performance when not using verification system on Tomcat is better than when using the WCVR system. As shown in Figure 6.12 the average response time through the WCVR was 5.362 seconds on Tomcat, while the average of response time when not using verification system was 2.386 seconds on Tomcat. When using the WCVR system to verify the dynamic web content, it degrades the overall performance with respect to the type of web content.

We also observe that the overall performance through the WCVR (on IIS and Tomcat web servers) is considerably improved as compared to that obtained when using the DSSA. This is understandable because we have utilised the performance of a WCVR using a new hashing strategy.

However, some common bottlenecks (such as network bandwidth, CPU, memory and I/O) occurred in our experimental studies, this led to an increase in the overhead of our proposed system. If the CPU utilization is over 80%, it will increase exponentially rather than linearly. In addition, our experimental results indicate that the importance of memory : the memory in our experiments was insufficient and this led to the disconnection of the IIS and Tomcat web server.

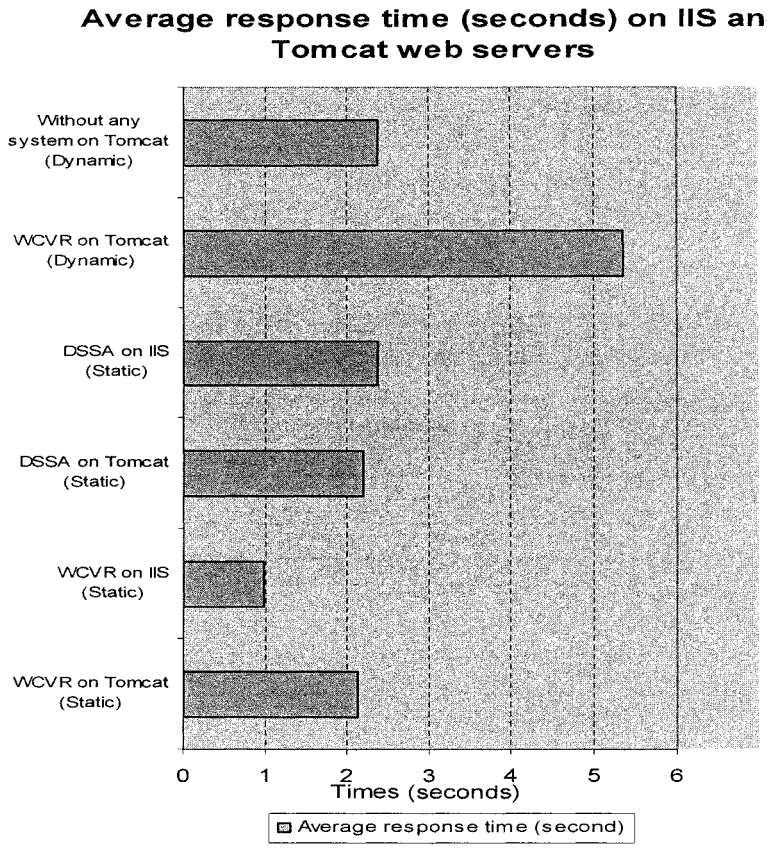


Figure 6.12: Average response time (seconds) on Tomcat and IIS web servers.

The content type, static or dynamic, will also have in impact on overhead. For example, server-side dynamic content, such as CGI or Servlets, make it difficult to size the processor. The processing costs of this dynamic content will be high enough so that the web server processing cost is not significant. This assumes that the web server has been correctly configured (133). Therefore, given more memory on a large scale web server, it is unlikely that the WCVR system would crash.

A comparison[4] of different web servers performance when generating dynamic

[4]http://www.pcmag.com/topic/0,2944,t=John%20Clyman,00.asp.

pages (using a Common Gateway Interface), illustrates dramatic decreases in a web server performance when generating dynamic pages.

Readers should note that longer keys in hashing take much more computing resource to decrypt, and hence make them less vulnerable to attack. However, the SHA1-extended is also more costly in performance terms (see Table 6.4 and Table 6.5), but this is the cost that legitimate users pay for higher levels of security.

In this chapter, five case studies are presented to evaluate the reliability and performance of the proposed WCVR system: case study - security objective (detection and recovery), case study for micro-benchmarking performance (SHA-1, and SHA1-extended) and case studies K, R, and J to measure the end-to-end performance through (i) the WCVR system, (ii) the DSSA system and (iii) without any verification system.

We have tested our system in an environment which is composed of two web servers: Apache 1.3.29 with Tomcat container 5.01 on MS Windows Server 2003 and IIS 6.0 on MS Windows Server 2003. The two web servers contain a copy of the target web site and a shopping cart application. Over 70 attacks were performed against the server-side generated static and dynamic web content security properties. We exploited different types of vulnerabilities that allowed for the modification of files in the designated directories of a web server. During testing, all the attacks launched against the web servers were detected and recovered by the WCVR system.

To conclude, results from a series of experimental tests (K, R, and J case studies) appear to suggest that the WCVR satisfies the performance objective for verification of integrity of server-side web content. We believe that the overhead of WCVR system can be significantly decreased, if the server technical speculation follows the current industry standards. The experimental studies indicate that the performance of the WCVR is dependent upon the CPU processing capabilities, the memory size and network bandwidth.

In the next chapter, we will summerise the contributions in this thesis and highlight the future work.

Chapter 7

Conclusions and Future Work

7.1 Research Summary

Chapter 1 has described the thesis problem, motivation, aims, objectives and contributions. Although current security mechanisms could provide security against unauthorised access to system resources, several security incident reports from emergency response teams such as CERT and AusCERT clearly demonstrate that the available security mechanisms have not made system break-ins impossible. In chapter 2, we presented the description of web content and the HTTP request-response model. We also concentrated on definitions and objectives of web security. As mentioned in chapters 1 and 2, data integrity has received little attention in information security research and technical security groups and communities.

In chapter 3, we identified tampering attacks on server-side static and dynamic web content. Furthermore, we surveyed the limitations, requirements, strengths and weaknesses of basic security technologies (e.g., SSL, firewall, digital signature and data validation schemes) and existing integrity verification systems. Our investigation has indicated that because server-side dynamic web content is not verified unauthorised tampering is a potential issue . To address this issue, we propose a novel system, called the Web Content Verification and Recovery (WCVR) system (see chapter 4). In chapter 5, this proposed system has been implemented

into three mechanisms: web register, response hashing and HTTP interface, using tools such as Java, Servlet and Filters and DBMS Microsoft Database.

In chapter 6, we conducted a set of experiential studies to address the security and performance objectives. The results of the experimental studies have demonstrated that the proposed system (WCVR) provides a high coverage of detection and recovery.

The reminder of this chapter is organised into the following sections: Section 7.2 provids conclusions of this thesis. In Section 7.3, the contributions of this research work are summarised. Section 7.4 discusses several avenues for further work, and indicates the limitations of our research.

7.2 Conclusions of Thesis

In this research work, we have focused on tampering attacks on server-side static and dynamic web content. Server-side web content can be tampered with by altering the style classes such as Cascading Style Sheet (CSS) and referenced objects (images, audio, video and other objects). In addition, source code of the web page can be tampered with by running malicious code that compromises a requested page before the client receives it.

Our investigation indicates that data integrity has received little attention in information security research. Furthermore, there is little published research in methods for testing web content integrity. The published research and technical communities in web security area are generally more concerned with cryptographic rules and algorithms. In an attempt to address this, our thesis focuses on the integrity of data. Data integrity refers to the trustworthiness of information resources, thereby ensuring that only an authorised client can alter the data – unauthorised tampering may result in incorrect or malicious web application behaviour. The main finding in our literature review is that unauthorised tampering is still a potential problem because dynamic web content is not verified. The question this thesis addresses is how the integrity of server-side static and

dynamic web content can be verified to detect and recover from tampering attacks before the client receives the requested page.

We have presented a novel security system called Web Content Verification and Recovery (WCVR) system, which attempts to address the security issues (violation of data integrity) that arise in tampering attacks. We have developed a client interaction model and a new hashing strategy. Furthermore, we propose a real-time web security framework consisting of a state protocol of web security policies and a number of components that can be used to verify the static and dynamic web content. Our solution is implemented as a prototype. This prototype consists of three mechanisms: web register, HTTP interface and response hashing. In this thesis, we have conducted a set of experiential studies to meet the security (detection and recovery) and performance objectives. As a result, tests indicate that the WCVR is able to reliably and accurately detect and recover from tampering attacks on the server-side static and dynamic web content.

It should be noted that our approach is applicable to many common kinds tampering attacks such as visualisation spoofing attacks and processing of all data types on the server-side.

7.3 A Summary of Contributions

The main aim of this thesis is to investigate server-side static and dynamic web content survivability using a WCVR system. The main contributions of this thesis are as follows:

7.3.1 A novel approach to the verification of server-side dynamic web content integrity

In an attempt to overcome the research problem, we have proposed a novel survivable system that could provide continued and correct services to internal and

external users, even though a web data manipulation problem may have occurred. The proposed framework includes a new state protocol to enforce a set of web policies, and a supporting software system to verify server-side static and dynamic web content before the client receives the requested page. This approach would add confidence in terms of users correctly accessing web services and displaying electronic materials on their web browsers.

As an integrity check, the proposed state protocol is similar to adaptive intrusion-tolerant server system (110). For each web resource whose integrity is to be checked, a checksum (hash value) is computed from a private key. To resist possible guesses by an adversary, the checksum is computed by applying a one-way SHA-1 hash function to the concatenation of the private key and the content to be checked. The resulting checksum is then compared with a pre-computed one. This is sufficient to check if a web content has been tampered with.

We have also formulated a new model to deal with every interaction element (both HTML and non-HTML objects) in a web page called Client Interaction Elements (CIE) model. A CIE model extends Offutt et al.'s (4) HTML Input Units (IU) model which is only used to HTML input units to create tests on the client for web applications that violate checks on user inputs.

A CIE model is formulated to provide the automatic extraction of parameters (e.g. HTTP transfer mode, user, data, type of interaction element, and server page) of the client interaction element. The automatic extraction method provides a simple method for automatically analysing a web site that contains a large number of elements and where each web page includes a large number of interaction elements (e.g. forms, frames, links, and HTML and non-HTML objects).

7.3.2 Improved performance in the verification of static web content

In the case of static web content, the results of experiential studies (K, R, and J) appear to indicate that the average end-to-end performance when using the WCVR system (on IIS and Tomcat web servers) is better than when using the DSSA system. As shown Figure 6.12 the average response time through the WCVR was 2.12 seconds on Tomcat and 0.977 seconds on IIS, whilst the average of response time through DSSA was 2.198 seconds on Tomcat and was 2.39 seconds on IIS. The WCVR results (through the WCVR) are lower than the response times through the DSSA on both web server applications.

We also observe that the overall performance through the WCVR (on IIS and Tomcat web servers) is considerably improved as compared to that obtained when using the DSSA. This is understandable because we have utilised the performance of a WCVR using a new hashing strategy.

Therefore, we have developed a new hashing strategy to describe hashing calculations for the referenced objects that are shared among the web pages of the target web site or the web application. This strategy applies for all web referenced objects that have been developed before use over the secure HTTP request-response model. This new hashing strategy to improve the hashing performance and minimise overhead times of the proposed WCVR system.

7.3.3 The development of the WCVR system to ensure the survivability of web content

The WCVR prototype is implemented into three mechanisms: web register, response hashing and HTTP interface mechanism. We carried out a set of experiments to address the security objective (detection, and recovery) and the performance objective. The results of a pilot study have shown that the proposed system (WCVR) provides a high coverage of detection and recovery.

7.3.4 Experimental studies to evaluate the reliability and effectiveness of the WCVR system

In this thesis, five case studies are presented to evaluate the reliability and performance of the proposed WCVR system: case study - security objective (detection and recovery), case study for micro-benchmarking performance (SHA-1, and SHA1-extended) and case studies K, R, and J to measure the end-to-end performance through (i) the WCVR system, (ii) the DSSA system and (iii) without any verification system.

We have tested our system in an environment which is composed of two web servers: Apache 1.3.29 with Tomcat container 5.01 on MS Windows Server 2003 and IIS 6.0 on MS Windows Server 2003. The two web servers contain a copy of the target web site and a shopping cart application. Over 70 attacks were performed against the server-side generated static and dynamic web content security properties. We exploited different types of vulnerabilities that allowed for the modification of files in the designated directories of a web server. During testing, all the attacks launched against the web servers were detected and recovered by the WCVR system.

To conclude, results from a series of experimental tests (K, R, and J case studies) appear to suggest that the WCVR satisfies the performance objective for verification of integrity of server-side web content. We believe that the overhead of WCVR system can be significantly decreased, if the server technical speculation follows the current industry standards. The experimental studies indicate that the performance of the WCVR is dependent upon the CPU processing capabilities, the memory size and network bandwidth.

7.4 Limitations of Research Work and Future Work

It should be noted that the WCVR system has some limitations. These limitations are related directly to our solution design and implementation.

In the solution design, we have used a set of work assumptions that may limit the functionality of our solution. For example, we have assumed that SSL protocol is confined to secure the communication channel between the client and server sides. However, this assumption may be invalid if the integrity of data in transfer is violated. To address this issue, we believe that our approach can be extended to derive all tampering attacks over the HTTP request-response model include data integrity in communication channel. Such an extension is planned as future work.

Another work assumption is that the data validation modules should be operated in the client and server sides. But this assumption may be invalid if an existing web application does not have a data validation modules on the server-side. Therefore, future work to develop a data validation module on a server is related to our solution would help to address this issue.

Optimisation of our solution implementation is another issue for future work. The WCVR system tested is not optimised. We believe that the optimisation of the conceptual design is possible. Such an optimisation would increase the effectiveness of the WCVR system.

Appendix A

Case Study - Security Objective: Table

The following table contains the the whole list of alerted web content that are compromised by running malicious web content manipulation software.

To assess how successful the WCVR system is able to detect, and recover from the unknown tampering attacks; students (K, R, and J) started to run the proposed web register mechanism. This mechanism takes the hashing measurements for every server-side static web content stored in the designated directories of the suggested web sites/applications hosted on Tomcat and IIS web servers. Note, the response hashing mechanism takes the hashing measurement for the generated dynamic web content during online transactions (when a user request a web resource).

Over 70 tampering attacks were launched against the designated directories of the suggested web sites hosted on Tomcat and IIS web servers. During the testing, all the attacks launched against the web servers were detected and recovered by the WCVR system.

Test 1 **Tomcat**

Number	URL	Altered Data	Result	Detection	Recovery
1	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/st_proj.html	Title changed to 'This is a different name' . stweb.gif altered in paint, colour of letters changed Changed Background to Yellow	Unaltered web page, recovery succesful	Yes	Yes
2	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/st_proj.html	2007-2008 now equals 'This' 2005-2006 now equals 'Is' 2003-2004 now equals 'A'	Unaltered web page, recovery succesful	Yes	Yes
3	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/Hachett_trees07/trees.htm	Amanda.html/Kiana.htm has been Deleted Amandahtml/vanessa.htm has been deleted Amandahtml/micheal.htm has been deleted	Unaltered web page, recovery succesful	Yes	Yes
4	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/Hachett_tre	Deleted everything between Html tags	Unaltered web page, recovery succesful	Yes	Yes
5	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/rodpen07/penguins_rodman07.htm	penline.jpg swapped around with hobopenguin.jpg Underlined and altered colour of Miss Rodmans Third Graders	Unaltered web page, recovery succesful	Yes	Yes
6	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/rodman_about_both/rodman_about_both.htm	Delted CSS from this file	Unaltered web page, recovery succesful	Yes	Yes
7	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/dummett_penguins06/main.htm	Deleted all of the table	Unaltered web page, recovery succesful	Yes	Yes

8	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/teeth_riddles_du/contents.htm	Replaced the multiple images titled tooth5.gif, teethnn.gif and tttt.gif with a comment displaying the following characters, 'There is no picture'.	Unaltered web page, recovery succesful	Yes	Yes
9	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/teeth_riddles_du/trevor/index.html	Font altered from style 3 to ariel Background changed to pink 'Mind boggling teeth riddles' was changed to testing static content	Unaltered web page, recovery succesful	Yes	Yes
10	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/give_mrs_early/main.htm	Changing table width from 700 to 900 Changing border colour to blue Changing table alignment to left	Unaltered web page, recovery succesful	Yes	Yes
11	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/give_mrs_early/html_give/olivia.htm	Editing the four images sky colour to black. Changing overall background colour to pink	Unaltered web page, recovery succesful	Yes	Yes
12	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/early_tree06/tree.htm	Deleted Hyperlinks and altered images so they have a big black line going through the middle	Unaltered web page, recovery succesful	No	Yes
13	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/pollution_movies05/main.htm	Changed the following link 'http://www.apple.com/quicktime/products/qt/' to 'www.northumbria.ac.uk'	Unaltered web page, recovery succesful	Yes	Yes
14	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/Hachett_trees07/amandahtml/natalie.htm	Changing centre image to a red circle	Unaltered web page, recovery succesful	Yes	Yes
15	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/st_proj.html	Deleted all chain images	Unaltered web page, recovery succesful	Yes	Yes

Test 2 **IIS**

Number	URL	Altered Data	Result	Detection	Recovery
1	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/st_proj.html	Delete created by Mrs Rodman third graders, plus link, Change date of 2004-2005 to 2012-2005	Unaltered web page, recovery succesful	Yes	Yes
2	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/road_to_rev_students/timeline_indexvertiical.htm	Put simly face on image, Changed image size width to 700	Unaltered web page, recovery succesful	Yes	Yes
3	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/animals_margolis06/main.htm	Changing Johnathon to Shearer, Changed table to 700, Edit animals on the football picth	Unaltered web page, recovery succesful	Yes	Yes
4	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/na_webs_mckenna/main.htm	Delete title, Delete beaded.gif	Unaltered web page, recovery succesful	Yes	Yes
5	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/st_proj.html	Backgroung color to vilote, Delete 2007-2008	Unaltered web page, recovery succesful	Yes	Yes
6	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/teeth_riddles_du/contents.htm	Edit image tttt.gif background to grey, delete the toothbrushes, Changed table width to 1000	Unaltered web page, recovery succesful	Yes	Yes
7	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/Hachett_trees07/trees.htm	Changed arbres07.gif by spraypainting over the file, Delete trees by Denise	Unaltered web page, recovery succesful	Yes	Yes
8	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/rodman_about_both/rodman_about_both.htm	Changed back2.gif by making pencil pink, Changed Charles link too norhumbria.ac.uk	Unaltered web page, recovery succesful	Yes	Yes
9	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/rodman_about_both/a/letter.htm	Changed a.gif so the man has a beard, and mad light in the windows and had a red door	Unaltered web page, recovery succesful	Yes	Yes

10	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/guer_pen_poems/penguins_main.htm	Delete all links and CSS	Unaltered web page, recovery succesful	Yes	Yes
11	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/early_tree06/tree.htm	Changed alligment for Samatha to left and changed font size to 24px	Unaltered web page, recovery succesful	Yes	Yes
12	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/dummett_penguins06/main.htm	Changed hobopenguin.gif so penguin has a baseball hat and delete the massive penguin logo	Unaltered web page, recovery succesful	Yes	Yes
13	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/am_rev_timeline_sample/sampler.htm	Changed crispus_attucks.jpeg size too 100 and delete the image source james_armistead.jpeg	Unaltered web page, recovery succesful	Yes	Yes
14	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/pollution_movies05/main.htm	Changed background color to blue and changed Everglades to monitor	Unaltered web page, recovery succesful	Yes	Yes
15	http://192.168.10.1/tree2/hillside.needham.k12.ma.us/cyberventues/rodpen07/html/juliachinstrap.htm	Changed the direction of the arrows in juliachinstrap_1	Unaltered web page, recovery succesful	Yes	Yes

Appendix B

Case Study for Micro-benchmarking Performance: Tables

This appendix contains the following two tables:

1. Table 1: List of all entries that had been measured using SHA-1 function.
2. Table 2: List of all entries that had been measured using SHA1-extended function.

Table 1: Using SHA1

File Name	File Size (byte)	Checksum	Time (1)	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
Signature.txt	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0		
shadi.htm	94	1272572422	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
hadi.htm	133	884912936	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
TryBean.java	341	-1600651456	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
Exp1Bean.java	342	-232798031	0	15	0	0	0	0	0	0	0	0	0	0	0	0	0		
Trai2Bean.java	343	204520273	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
JsptrailBean.java	346	-1677907428	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0		
SetcookieBean.java	347	-2042327349	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
CookieformBean.java	348	1442310949	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
ShowcookievalueBea	353	-989928715	0	0	0	16	0	0	0	0	0	0	0	16	0	16	0		
CustomerRegistration	358	1154350516	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
LoginSession.inc.jsp	361	1373536135	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
ThankYou.java	366	-1343059341	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16		
Bean1.java	409	871613630	0	16	16	31	0	0	16	16	0	16	0	16	0	0	0		
travel-styles.css	424	-17980562	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
MainTop.inc.jsp	458	2108878059	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
TryBean.class	466	1401421774	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
Exp1Bean.class	468	1177154182	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
Trai2Bean.class	470	-408911572	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
JsptrailBean.class	476	1526137390	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0		
SetcookieBean.class	478	-1775340616	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
CookieformBean.clas	480	-1955559726	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0		
ShowcookievalueBea	490	-540965340	0	0	0	0	0	0	16	16	0	0	0	0	0	0	0		
CustomerRegistration	500	-191227066	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
WorkreportTop.inc.jsp	517	-859478096	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
IllegalRequest.jsp	556	1003298536	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
ThankYou.class	556	1131066933	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
StartHere.jsp	559	147946009	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
Bean1.class	564	-190535789	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
FilterServletOutputStr	579	-54711483	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
SimpleExample.jsp	580	-583901640	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
ThankYou.jsp	595	-1968407935	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
FindHotels.jsp	620	696647692	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0		
accounts.jsp	622	-1669476439	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0		

EditAccounts.jsp	625	245083767	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
RentCars.jsp	630	2000656234	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0		
GenericServlet.java	650	1608865819	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0		
SocketUtil.java	683	-457906586	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0		
SimplePrimeExample	689	2095463072	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
FilterServletOutputStr	704	1887981710	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
GenericServlet.java~2	720	-1086882945	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
PrimeExample.jsp	743	696948914	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
LogOut.jsp	765	-1028653302	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
Store.jsp	765	-612654884	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
IllegalHeadingExamp	797	-161636142	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
SocketUtil.class	827	-1952386898	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0		
RepeatExample.jsp	856	1870225064	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0		
BoardNoticeDelete.in	880	-1653066870	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
JSP-Styles.css	888	-1867757700	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
DebugExample.jsp	904	-892866257	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0		
GenericServlet.class	908	-1106515258	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
BoardPage.inc.jsp	970	349991388	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
BoardPageAdmin.inc	970	349991388	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0		
airplane.gif	975	131228564	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
[redacted]																			
MainView.jsp	1097	510895866	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
WorkreportDeleteAdn	1097	-759682113	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
InfoDeleteAdmin.inc.j	1099	-1394978379	16	0	0	0	0	0	0	0	16	0	15	0	0	0	0		
dollar.gif	1100	97843809	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0		
car.gif	1147	-1420249025	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16		
Copy of car.gif	1147	-1420249025	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0		
BookFlights.jsp	1174	-945255418	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0		
IfExample.jsp	1203	1792013163	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
WorkreportContent.in	1226	-1709436867	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0		
HttpClient.java	1317	1868499907	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
larrow.gif	1332	-1371724785	0	16	16	0	0	0	16	0	0	0	0	0	0	0	15		
WebClient.java~1~	1338	-2017247515	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

WebClient.java~2~	1340	387948838	0	0	0	0	0	0	0	0	0	15	0	0	0	0	0		
WebClient.java~6~	1340	1639038175	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0		
WebClient.class	1341	1079977277	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0		
WebClient.java	1342	676691846	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0		
WebClient.java~3~	1342	676691846	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
WebClient.java~4~	1342	-141169945	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16		
WebClient.java~5~	1342	676691846	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0		
GenericResponseWra	1355	-139620875	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
BoardNoticeView.inc.	1367	872389387	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
rarrow.gif	1380	-25496505	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
web.xml	1430	-29429241	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
InfoChangeAdmin.jsp	1440	-1465229931	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
WorkreportModify.inc	1450	2144877508	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
WorkreportMy.jsp	1476	887760261	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
MainMenu.inc.jsp	1499	-1178118973	0	0	0	0	0	16	0	0	0	0	0	0	0	15	0		
passport.gif	1557	-871658710	0	15	15	0	0	0	15	0	0	0	0	0	0	0	0		
ByteArrayServletOutp	1559	255888838	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0		
art.jsp	1589	1705820666	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
Cart.jsp	1589	1705820666	31	0	0	0	0	0	0	0	0	16	0	0	0	0	0		
FilterExample.jsp	1610	980958151	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
bed.gif	1668	121687676	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
MainMenuAdmin.inc.j	1669	-988530857	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
NetworkClient.class	1746	605566764	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
BoardLineDelete.inc.j	1760	-1103458797	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16		
NetworkClient.java	1760	-1108022713	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
Purchase.java	1782	-275538599	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
airplane-large.gif	1812	-954843052	0	16	16	0	0	0	15	0	16	15	16	16	0	0	0		
integrityfilter.txt	1846	627197437	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
HttpClient.class	1909	-77188432	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
BoardNoticeConfirmA	1918	1462315427	0	0	0	0	15	16	0	16	0	0	0	16	15	16	0		
quick-search.html	1959	-1641397241	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0		
DBConnection.class	1966	346487916	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
OutputCaptureRespo	1970	1719357629	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
BoardNoticeView.jsp	2009	1190396413	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
Purchase.class	2025	-1457325174	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0		
Join.jsp	2047	622822035	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
Signature.class	2049	435310626	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0		

BoardNoticeConfirm.j	2121	1046499632	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
original_response.jav	2168	-1158740420	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0		
original_response.jav	2170	1058978920	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
WorkreportModify.jsp	2172	2028437784	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
original_response.jav	2177	-365499177	0	0	0	0	0	0	0	16	0	0	0	15	0	16	0		
interceptor.class	2190	2105160438	0	0	0	0	0	0	0	0	0	0	0	0	15	0	0		
Purchase.jsp	2206	1367895750	0	0	0	0	0	0	0	0	0	0	0	0	15	0	0		
original_response.jav	2218	842483803	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
original_response.cla	2225	-844775128	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
favicon.ico	2238	-358074893	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
BoardNotice.jsp	2244	-797211816	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
tomcat-power.gif	2324	232558536	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
BoardNoticeAdmin.js	2334	1654751691	15	0	0	0	0	0	0	0	15	0	0	0	0	0	0		
Store.java	2349	-719748037	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0		
ReadMe.txt	2398	951290428	0	0	0	16	16	0	0	0	0	0	0	0	0	0	0		
FilterServletOutputStr	2435	342175936	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
OutputCaptureRespo	2435	342175936	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
OutputCaptureRespo	2435	342175936	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
WorkreportProcess.in	2480	-373706408	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0		
Store.class	2501	1955581330	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0		
WorkreportMyAdmin.j	2523	172014028	0	0	0	0	0	15	0	0	0	0	0	0	0	0	0		
InfoAllAdmin.jsp	2566	1881608722	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15		
InfoModifyAdmin.jsp	2601	-850667950	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
mainfilter.class	2739	-1372143660	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0		
tomcat.gif	2767	1534475753	0	16	0	0	0	0	0	15	16	15	15	0	0	0	15		
tomcat1.gif	2770	2106935856	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
interceptor.java~5~	2797	539216908	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
interceptor.java~6~	2823	-1622001508	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
index.jsp	2835	1343950008	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
interceptor.java~7~	2857	323719677	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
interceptor.java~8~	2897	-1807916754	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
Join.inc.jsp	2911	-784451897	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
interceptor.java~10~	2928	616585041	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
interceptor.java~9~	2928	-34512159	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
interceptor.java~11~	2936	-1152734193	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15		
interceptor.java~12~	2937	1621582348	16	0	0	0	0	0	31	0	0	0	15	0	16	0	0		
interceptor.java~13~	2977	-497137884	0	31	0	0	0	0	0	0	0	0	0	0	16	0	0		

InfoMy.inc.jsp	3014	1849982161	0	0	0	0	0	0	0	0	0	0	0	15	0	0	0		
InfoModifyAdmin.inc.j	3048	231662408	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
Cart.class	3077	-447479607	0	0	0	0	0	31	0	0	0	0	0	0	0	0	0		
DBConnection.java	3085	-1302483155	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
GenericResponseWra	3134	-1628104902	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
original_response.jav	3134	-1628104902	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
original_response.jav	3134	-1628104902	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
original_response.jav	3135	-1599864127	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
mainfilter.java~2~	3168	1594995429	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
mainfilter.java	3172	112181547	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
mainfilter.java~3~	3173	-2038353487	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
interceptor.java	3186	-1240385344	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
interceptor.java~14~	3191	1621075004	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0		
original_response.jav	3269	-528300110	0	0	16	0	0	15	0	0	0	0	15	0	0	0	15		
original_response.jav	3318	210988333	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16		
original_response.jav	3318	210988333	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
original_response.jav	3364	2088672851	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
WorkreportAdmin.jsp	3471	725447933	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0		
BoardLine.inc.jsp	3497	1288353540	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
index.inc.jsp	3517	-626865474	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0		
Cart.java	3567	-356039272	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
InfoMy.jsp	3700	305702591	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0		
csajsp-taglib-short.tld	3813	-328499648	0	16	0	0	0	0	0	0	0	0	0	0	0	0	16		
Copy of Signature.jav	3897	-32121694	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
GenericServlet.java~1	3897	-32121694	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
integrityfilter.class	3897	1319325810	0	0	0	0	15	0	0	0	0	0	0	0	16	0	0		
Signature.java	3897	-32121694	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
[redacted]																			
greenpaper.jpg	4498	258146265	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0		
integrityfilter.java~105	4732	-1827739517	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
integrityfilter.java~106	4734	-2040608029	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
books.gif	4750	-516374921	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
integrityfilter.java~107	4785	1287809676	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

integrityfilter.java~108	4787	2052019713	0	16	31	16	0	16	16	0	0	15	0	0	0	16	0		
integrityfilter.java~109	4842	1746762139	0	0	16	0	0	0	0	0	15	0	0	16	0	0	0		
integrityfilter.java~110	4898	-1741831420	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
integrityfilter.java~111	4902	1722012420	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
integrityfilter.java	4912	-911101049	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
WorkreportList.inc.jsp	4960	-696389675	0	15	15	0	0	0	0	0	16	15	16	16	0	0	16		
csajsp-taglib.tld	5279	351207873	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0		
jakarta-banner.gif	8584	-157659447	0	0	0	15	0	0	0	0	0	15	0	0	0	0	0		
RELEASE-NOTES.tx	8999	-1497947984	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0		
index.jsp	9367	1294681762	0	0	0	0	0	15	0	15	0	0	0	0	0	15	0		
Copy of greenpaper.g	11387	-744658804	16	15	0	0	0	16	0	0	0	0	0	0	0	16	0		
greenpaper.gif	11387	-744658804	0	0	0	16	0	0	0	16	0	0	0	0	0	0	0		
schema.sql	17281	-626797141	0	16	0	0	0	0	0	0	0	16	0	0	0	0	0	2.133	
gims.tar.tar	19406	-813173124	0	16	0	0	0	0	16	0	0	0	0	0	0	0	0		
Thumbs.db	40448	-1765323596	16	0	0	0	16	0	0	0	16	0	0	15	16	0	0		
Thumbs.db	64000	559923886	0	0	0	16	16	0	0	0	0	0	0	0	16	0	0	3.2	
[redacted]																			
[redacted]																			
[redacted]																			

Table 2: Using SHA1-extended

Checksum	Time (1)	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
E9ED3C0BE66BCD6D7A0C8200FE991DA449E00AFF	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0		
5617516999AE90ADB0E5DEDA75F8DD0C5CB9EACB	0	0	0	16	0	16	0	0	15	0	0	0	0	16	0		
DCDA5C17975E651EEC0280BDD4ECD246D1161D98	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
4A138301E2F36ED7C1575BAFB50FA367169EF3A2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
B7EA3D21D60B004B64C6A7B1CE4B94F8AE82FF52	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0		
18013E03918F105A6C1C88159D61C4E86070A20F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0F79E5EE701992C0282EE39D9BEA4B6B43D80982	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0		
12877244D7B485D3FE1CD07B4EE96C327580F3AF	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
08B016C760F7E63EDEC0FBF862EE75120A15C373	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
2478DFEFCFA0AF599FE7AC9DEFCBA333EA5CB295	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0		
19F6E6F6ED3F0196623C1939BF14A1BF131B8495	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
C6D88257270724C61EF6D860B71BBDF5E153065E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
56F2A1BBC4AF652CF9DAADAF7C9BC4D8AD1C9C8A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
9D869512F6747FE351B8D825F43BF506562CB0BE	0	0	16	16	0	0	0	0	0	0	15	0	15	0	0		
8C967D27FB403E39848F46563070046EDA501529	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0FFB9CF301BD2C74765846162C0ED8B489988AA9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16		
D24ACB5B73F43866A85A27EC69E4A20B218ED9C1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
B7B2B14E9D919600B3721DFAF23B3CED90B3EA49	0	0	0	0	0	0	0	0	0	0	0	15	0	0	0		
30B3683E06B9C669E32C9AFE9A25BD71DA154BE3	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0		
233E99CD6168386FEDE3C2A663F378CB666DDAAC	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
705B976F79680E548CC58E8B6E4E2850F830EA39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
DB101DFE3FABCDDF35F4D943ACBA9218EB29EF6D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
04EF4FCAD78170379021CCD808FCE376D0C52DC5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0E678ECBD7D811FDC01D935AF790C0E43DED0C02	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0		
8193CD694297EA9F695D9BA02E1AD783C514CCD9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
D8611E80F79C5D9A6DDC2C5C9163C6323726DBF1	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0		
765694B539407D3954A175E87229218439B00068	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
56236652C0E32364638C5294B501E786BD0F4B91	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
27FAEB8E7395BEDFE3A84ED7BFDB1F91036131BA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16		
27B3DC68480618CD62DE2696FED0779E4ED96F77	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
9BC88EAC703BBE846ED2F9826C2554A671A8094B	0	0	0	0	0	0	0	0	0	0	0	0	15	15	0		
FDD31E21F80E4DFD1B6AFD2BA0951840EE2A2C7F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
C146240D42A0B3250C8C08B824CA47260822ABE0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
9D354FF4B08DDB0FE8BD0656692AE895C6F36887	313	312	328	312	312	312	312	313	313	313	313	313	328	422	313		

382F3100DB971F8DA24FF61C8B3669812ABE1008	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16		
3624F604FA6DBF02B4335E90A00B67F392160EEC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
400DB2971E6767F9DEDF2556C7300EE4DF2E9388	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
43996E7F1AE3A6F5B73B28C6773350AC7F5D9ACF	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
952FFD4039B3F2FC9CFE13D9A8C30824FAC24A93	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15		
33C332ECBE7CC0AC2DB682A67156B2DBEDD4FC8A	0	0	0	0	0	0	15	0	0	15	0	0	0	0	0		
98BC3697BD166BFE50171D3AA7FB4440868FA90A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
8FB8FC870669AD11E60600767B1494F9AE29F0D5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
06B5679A00D21448D0F4295CD9AA2FD9FB347E7B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0709306BA800150FB58C6520F3310AEBD759AA16	0	0	0	0	0	0	0	15	0	15	0	0	0	0	0		
88BEE2C00748323CABC989819A9FB523EC0BD6DB	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
72B5FA58D2BE1531199EC02BD87DBC46B1E9CC30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
CD9A957DD00BC6E61F7FAA287E8896114B940BC4	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0		
F207261233DF86A0038BF39B6FF82BC1F3E61347	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
B495285568467A25CCB322B6C3AFA5EA0D05954E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
4083047D77D35CA012148DB20B3D5FB2BC855F3E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
F462937F7E33B89BCCA754E7DC7A216D2932AB1D	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0		
7F9BE9303606626BBD86622927EA48F95BDF8C54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
7F9BE9303606626BBD86622927EA48F95BDF8C54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
45A9CF81C355F5383E9CD18C3F7BDDFDB1062003	0	0	0	16	0	0	0	15	0	0	0	15	0	15	0		
[redacted]																	
5717A151C0096D0D4F8DE6CA1BBF103FF0F77C5C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
FDB112F873263E722FE5945C8640DC7E9313B2B7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
5A12B34A77AA8CF4A5E7A3E202DCA026CEC9D683	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0		
FA6E920A095B44E4635A527436703B7FF6AB2006	16	16	16	15	16	16	15	16	16	16	16	16	16	16	16		
096BE0CB7EF151350329EB96AD1C9A103D0EC2B8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
096BE0CB7EF151350329EB96AD1C9A103D0EC2B8	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0		
9F070061EEF2A97B2C358BE17A0727B9774D2666	0	0	0	0	0	0	0	0	0	0	15	0	0	0	0		
DADACDAF75744168DF0E9F81C8C7DC384BFA2EA8	0	0	16	15	0	0	0	16	0	16	16	16	16	0	0		
41C527862135CBB9C3E8548FCB66F56C3AB5B7C1	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0		
0C76D7AFFE003E93B33A6DA1772A23AFA10DAC86	0	16	0	0	0	16	0	0	16	0	0	0	0	0	0		
01F95F974F0873D8A91EB0F98288E108F1C741E1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
36BC2B34C4CDD63F0CB0CDDA8846F865C3635EBC	0	0	0	0	0	0	0	0	0	0	16	16	0	0	0		

EB4EA6C038EF92930D1F36498FC4A3628296EB85	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
7910FF47626E9D9A84B93F371B15BB87A6713A54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
F14E2FBC290434515D2EDA5BDF82C2B73CED9E1E	0	0	16	0	0	0	0	16	15	0	0	0	0	0	0		
196A8FCF0B2ADA063DA886859D2C9DBFC0688495	0	16	0	0	0	0	0	0	0	0	0	0	15	0	0		
196A8FCF0B2ADA063DA886859D2C9DBFC0688495	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0		
52F5FE872CD48F9B28CFFA4E4F6A91FD1950A188	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
196A8FCF0B2ADA063DA886859D2C9DBFC0688495	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
C8D11FBAE84495C0DB9A5B5CE9CA3D7968168B05	0	0	15	15	15	0	0	0	0	0	0	0	0	0	0		
9D96ED4C718E2CC20850E01281E6309837DD458E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16		
5CC69AB3A35B0A38092BB44FDA86CABE05902202	0	0	16	0	0	0	0	0	0	0	0	0	0	0	15		
56177ADE7570E9BF0B55F5FD58DE64AE3BF1ECBE	16	0	0	0	0	0	0	0	0	0	0	0	0	0	16		
D4E54E5434E6A017168AD66E114B60D48BB91635	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0		
A5DFB5BB263CE3392F8FE3BEC471105D28D2D3AB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
E9A2C6CDACCB8930140F74DB78AD56768D549692	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
BCC657C15E2CBB2DDEE9DD795D891440AD36CC29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
1BC2E837168DC164590611744E9E1FC66AAD35AC	0	0	0	0	0	0	0	0	0	0	15	0	0	0	0		
F31B8C0CDCD7B842A4E6A439B1ACA91B6268206C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0887CBC41B81A9C418EF1ED91C5AC4D4FA93D14A	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0		
0887CBC41B81A9C418EF1ED91C5AC4D4FA93D14A	15	0	0	0	0	0	0	0	0	0	0	0	16	0	0		
43C7F207CAE8E2750DDD25042F7C929F6742ACA9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
F688117B8752340851B89ECA5ECC853915815FB8	0	16	0	0	15	16	0	0	16	0	0	0	0	0	0		
7C90479C78619DAD8F0B356F401C13C47DF68DA4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
D9871A9D6EE94EAFCAFB9A909D0FA57B177653D3	0	0	0	0	0	0	0	0	15	15	0	0	16	0	0		
0FEE9BF2DBF8B1FBE0A676E7DB0652E0EA78E93E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
01EC9F971B4F24E11484A317E0636E05424E7A7A	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0		
E6DC67E494A62D9DCADFB34CF77016061E87D7C7	0	15	0	16	0	15	0	0	16	0	0	0	0	0	0		
3AC2737D5DBAEBA408E3513AF402158ACE0A4BE7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
5A8E0D64A403EF4087A05D0466978C2539B674A5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
C158B0397AE14D2E39E829642BA52C636F767CBC	0	15	0	0	0	0	0	0	0	0	0	0	0	0	0		
95EF9CDD5A333A2BB394CF7031BC042E0F70D920	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0		
21FA614A4779A25EDEC9847049873531B9D29823	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16		
C063AE79D97BA8B055DDD99C559D0FF326FA3428	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
D8A88F5684D1178801D1C23A07165A18A8066EFA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
E02AA7582CDAAA97D03D11F1CB7EF5DEC48A6109	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
F43865157BCA75C35A5BB8F25A7DF1B228E7D090	0	0	0	0	0	0	0	0	0	16	15	0	0	0	0		
4ED05BDB227CECCE7A48106E86246CD3957C13D5	0	0	16	0	0	0	0	0	0	0	0	0	15	0	0		
E3103A704788F53527AAE690DA3FBF8431AFD25E	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

8423D3C2F33DF9D9A974225C8BFCEB21AA2D744A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
70DE53CD1E7709EE793F77CB5709C5265197CDDF	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
63BBD35F88D375C04B675C01698BBC0C9AAAA164	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
EE21D3F5C2DAE16D7BA2ADAA7F6EBCE5E486FD37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
80FB33C148C4E6443F8683DD2E59131AD294A0B2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
15C6EB46757270B29B6D3D2CA475C87BDBBE2238	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
AC9180CF4163B61531C2DAFDCCDFA5D0B3D20387	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
E2E63517D1671494B214C060B39E1AAA0256043B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
6635B60594DA852287F03191BB1A830C99592A5D	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0		
389AAD5BFFA0EE8B6A0A69D64A7BAB78E8694EB6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
E05BD6A195DE6E6F4D978E1ABAAB2502DC1DAC3A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
1E22C3B19776413F3A18CCCE033B681A1727E5AF	0	15	0	0	0	0	0	0	0	0	0	0	0	0	0		
6984E5C7B5692D3D6258DED2A4BACC082ED3EECC	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
ACA64644098A2C1A069749B561053240AEABE256	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0C702BC3E5C569CE09A672E92CBB4F9E3B83B412	0	0	0	0	0	0	0	0	0	0	0	15	0	0	0		
7CB50EA96C9C856947CD62375A62FC1F3A198539	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
7CB50EA96C9C856947CD62375A62FC1F3A198539	0	16	16	0	0	16	0	16	0	0	0	0	0	0	0		
7CB50EA96C9C856947CD62375A62FC1F3A198539	0	0	0	0	15	0	0	0	0	0	0	0	16	0	0		
239AFA14B6F4E657C14FAA1A48D53BF156169BA8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
63F42505E6F4471D230CDFCC5786980A29A2AD8F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
45C7477DAD771AE9ADACE222D3BC1490647E0DF8	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0		
455B0E85605B3C0537830FB84ED1B097528C07FD	0	16	0	0	0	0	31	0	16	0	0	0	0	0	0		
06415A3645A4813FE458D580B130E36F0CD9844B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
DE5026893BC3D9821C00F933F779DD255F0E5736	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
39E68A9BB7F76384243E913C0A44F8AC93BB51ED	0	0	0	0	0	0	15	0	0	0	15	0	0	0	16		
7ED1C4BB768D1BC832F66BB071A3E8071E3EFE64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
17B14A0D01D8CF437AC7C14C2D39A099E201828F	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0		
C359A88BBA362FD17E6676CDEDD8DEB204007F0E	0	0	16	0	0	0	0	0	0	0	0	0	0	15	0		
B69A0108D58EEA2D34562CA0CB211757DBB1CCD3	0	16	0	0	16	16	0	16	0	16	0	0	0	16	0		
7E085CC0813DA267195F874F2532213DC662D63B	0	0	0	0	0	0	16	0	0	0	0	0	15	0	16		
A301E318DADB7C8192F2EAABA728A3FE30095B3D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
DD72CE3812CB1CBF32875E18516A667DB831B6C0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
5253F5ED798718D1FB9C70E1A246E98C5AD6E05D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
B6EE55041467C1F57117E247E208A0AEF4A52CF3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
30406EE14CEDB4F9C444D949F4FC14BA4F1E61EC	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0		
76382E58D0EBC461664308A5A258425D2615985C	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0		
FFC10C0AFF62766265C850C042A69930D75654E0	0	0	0	0	0	0	0	0	0	16	0	0	0	15	0		

68154E01F6D8969DBE22C4B9F250D199563D259A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
924FD07D1691BED06E1730DAA054B80A5EB52571	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
DFFFA7F8609B59F5F555989DDD404B319ABB0331	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
1BC667A7899C3469C51F8B085379D61FF7E11630	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
90102510AB1635745139AE3355529AEA09A5DE6A	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
90102510AB1635745139AE3355529AEA09A5DE6A	0	0	0	0	0	0	0	0	0	0	16	15	0	0	0		
90102510AB1635745139AE3355529AEA09A5DE6A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
7BC32D3327496BF88A8699C752CF5343AFB49C15	0	0	0	0	16	0	15	0	0	0	0	0	0	0	0		
97BF6DC687D0A19496BB6222CEDFD867C7FA1575	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
A66EE0077878C506588E1B271672EF6C2F128226	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
2F8DE23F62C1878F38322BE0C9E90025CFD1BB0D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
ACBFAD0588250F3697F9B94D0E49039E99063FC8	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0		
C1E9F8FC918A525771EE7340A77035445BCA90A2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16		
908E515F9287F46CE50C5095BD77ED4499E72B19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0AE07AFC67E926B5FF5B8A15FD77BEE09BB075B4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0AE07AFC67E926B5FF5B8A15FD77BEE09BB075B4	0	0	0	0	0	0	16	0	0	0	0	0	0	0	15		
3AE3393C3616C22EC44A595995073875082B106D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
BFE919C433A9C46E3C1272BE8BF465124B385022	0	15	0	0	0	0	0	0	0	0	0	0	0	0	0		
9B046F66E12BB357AA4B2DE3464AD604F79ECB49	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
AE6D003B83C57432996D126137CD2B3D615EF8DC	0	0	0	16	0	0	0	0	15	0	0	0	0	0	0		
CB04358E629B33303E0E85CCF6A35701770999EC	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
F4BD0F67238DF8059583C4A26FDEDC319FA52452	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
03C5A2CAB07CDE33244AF228F94CE7CF80157033	0	0	0	0	16	0	16	0	0	0	0	0	0	0	0		
736CAF04CBC9359D7626C032DFC90D3FE5DB4EF1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
736CAF04CBC9359D7626C032DFC90D3FE5DB4EF1	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0		
3CB0698C702B8C99386C813D2732B22495F079A5	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0		
736CAF04CBC9359D7626C032DFC90D3FE5DB4EF1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16		average
0F22080	0	15	0	0	0	16	0	0	0	0	0	0	0	0	0	2.0667	2.533
CD0D61	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	2.0667	2.533
2216	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	1.9333	2.4
1772	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	2.0667	2.533
00796D	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	2.1333	2.667
7A9446C726E0A4C846A9C025A0289BE0282362E0	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
F30FA15A440906AD0C4FAB3F2504FC7363C4CF88	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
15A76234A8ED718E7E49BBBEB256DF7BBCB5FB08	0	0	16	16	0	0	0	0	0	0	15	0	0	0	0		
128A2F79166949CB1DAB5A5708555FDE308AE77B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
A13614EA6EA0658719AD3ECC7A9D7B9B2F4705C4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

D8317E2F55301821155BCC5F25416A76D5D630DB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
7056AE442D75E103C177318BE25F06F8F09124EC	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
56DAA5B978A8591A6E1F9FEC746D5A7219DF1EA8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
20D519A06095F9000CDF6E7B48A7D60FC7F7272F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
B083BE374D9815E6582969A635A8D2559B0567D0	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0		
4D87E41A9E46CA3B294D375CB013DF7E40FB11E6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
8C4B059D09CA093295A21A98C91F2075011E952E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0E35245AF3DFF49FD7AA7D3A27C85F0508635681	0	15	0	0	15	0	16	0	0	15	0	0	0	16	0		
C018BEBEEE720FE5658B059A874F1CC377CFCEF8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
E68139D93CC0EA8DA4C98110258D431A62F8F236	0	0	0	16	0	0	0	15	16	0	0	0	0	0	0		
6437C63F2A3FDEFE75588BEB03E565E455F52976	0	15	0	16	16	15	16	16	15	15	0	0	0	0	0		
6437C63F2A3FDEFE75588BEB03E565E455F52976	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0		
[redacted]																	
1B993C1218C8261387C55DE17741D96B863501B3	0	0	0	0	0	15	0	0	0	0	0	16	0	0	15		
9399C15E23CB0C237E8CCACBCCD801EAE5FA5612	0	0	0	0	0	0	0	0	0	0	0	15	0	15	0		
0FB728BBC5D11C2F27215EE80D97E72DF233F1FC	0	16	0	16	31	31	16	15	31	16	0	15	0	0	0	12.467	
[redacted]																	
[redacted]																	
[redacted]																	
[redacted]																	

15640
2188

References

[1] Chang, C.C., Hu, Y.S., Lu, T.C.: A watermarking-based image ownership and tampering authentication scheme. Pattern Recognition Letters **27**(5) (2006) 439–446 1, 4, 13

[2] Wu, H.C., Chang, C.C.: A novel digital image watermarking scheme based on the vector quantization technique. Computers & Security **24**(6) (2005) 460–471 1, 4, 13

[3] Rey, C., Dugelay, J.: A Survey of Watermarking Algorithms for Image Authentication. EURASIP Journal on Applied Signal Processing **6** (2002) 613–621 1, 4, 13

[4] Offutt, J., Wu, Y., Du, X., Huang, H.: Bypass testing of web applications. In: ISSRE 2004 15th International Symposium on Software Reliability Engineering. IEEE Computer Society, Los Alamitos, CA (2004) 187–197 1, 2, 5, 13, 50, 51, 61, 64, 71, 72, 74

[5] Calabrese, T.: Information Security Intelligence: Cryptographic Principles & Applications. Formely Thomson Learning (2004) 1, 13

[6] Song, Y., Beznosov, K., Leung, V.: Multiple-channel security architecture and its implementation over SSL. EURASIP J. Wirel. Commun. Netw. **2006**(2) (2006) 78–78 Hindawi Publishing Corp., New York, NY, United States. 2, 5, 13, 45, 46, 47

[7] Sedaghat, S.: Web authenticity. Master's thesis, University of Western Sydney, Australia (2002) 2, 3, 4, 5, 15, 23, 28, 29, 33, 47, 48, 50, 51, 57, 61, 64, 67, 72, 74, 81

[8] Gehling, B., Stankard, D.: eCommerce security. In: Proceedings of Information Security Curriculum Development (InfoSecCD) Conference 05, Kennesaw, GA, USA (2005) 32–37 2, 5, 26, 27, 28, 34, 40, 41, 51, 61, 64, 67

[9] Hassinen, M., Mussalo, P.: Client controlled security for web applications. In Wener, B., ed.: The IEEE Conference on Local Computer Networks 30th Anniversary, Australia, IEEE Computer Society Press (2005) 810–816 2, 5, 20, 56, 61, 64, 67, 69

[10] Probert, R., Stepien, B., Xiong, P.: Formal testing of web content using TTCN-3. In: TTCN-3 User Conference 2005. (2005) 2

[11] Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and implementation of a TCG-based integrity measurement architecture. In: USENIX Security Symposium. (2004) 223–238 2, 4, 61, 64

[12] Sedaghat, S., Pieprzyk, J., Vossough, E.: On-the-fly web content integrity check boosts users' confidence. Commun. ACM **45**(11) (2002) 33–37 2, 3, 5, 47, 48, 51, 57, 64, 65, 67, 81, 120

[13] Gritzalis, S., Spinellis, D.: Addressing threats and security issues in World Wide Web technology. In: Proceedings CMS '97 3rd IFIP TC6/TC11 International joint working Conference on Communications and Multimedia Security, IFIP, Chapman & Hall (1997) 33–46 2, 5, 14, 18, 20, 22, 34, 44, 45

[14] Reis, C., Dunagan, J., Wang, H., Dubrovsky, O., Esmair, S.: Browsershield: Vulnerability-driven filtering of dynamic HTML. In: Proceedings OSDI 06 7th USENIX Symposium on Operating Systems Design and Implementation. USENIX Association (2006) 61–74 2, 42, 61, 64, 72

[15] Sailer, R., Zhang, X., Jaeger, T., Doorn, L.: Design and implementation of a tcg-based integrity measurement architecture. In: SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium, Berkeley, CA, USA, USENIX Association (2004) 16–16 3, 41, 42

[16] Stekelenburg, M.V.: Integrity: using the undefined. In: BILETA '21: Proceedings of the 21st British & Irish Law, Education and Technology Association. (2006) 1–15 4

[17] Glisson, W., Welland, R.: Web development evolution: The assimilation of web engineering security. In: LA-WEB '05: Proceedings of the Third Latin American Web Congress, Washington, DC, USA, IEEE Computer Society (2005) 49 4, 26, 27, 28, 33, 55

[18] Ye, E., Yuan, Y., Smith, S.: Web Spoofing Revisited: SSL and Beyond. Technical Report TR2002-417, Dartmouth College, Computer Science, Hanover, NH (2002) 4

[19] Awad, E.: Electronic Commerce: From Vision to Fulfillment (3rd Edition). Prentice-Hall, Inc., Upper Saddle River, NJ, USA (2006) 4

[20] Rubin, A.: White-hat security arsenal: tackling the threats. Addison-Wesley Longman Ltd., Essex, UK, UK (2001) 4, 67

[21] Rubin, A., Geer, D.: A survey of web security. IEEE Computer **31**(9) (1998) 34–41 4, 13, 39, 40, 67

[22] CERT: CERT statistics 1988–2006. http://www.cert.org/stats (2006) 4

[23] Dierks, T., Allen, C.: The TLS Protocol Version 1.0 (1999) RFC Editor, United States. 5, 6, 44

[24] Oppliger, R., Gajek, S.: Effective Protection Against Phishing and Web Spoofing. In: Communications and Multimedia Security. (2005) 32–41 5, 34, 107

[25] Bass, T.: CEP and SOA: An open event-driven architecture for risk management. IT Financial Services '07, Portugal (2007) www.idc.pt/resources/PPTs/2007/Financial_Services/7_TI BCO.pdf. 5, 7, 48

[26] Bingyang, Z.: An Integrated Web Security System. In: DEXA '03: Proceedings of the 14th International Workshop on Database and Expert Systems Applications, Washington, DC, USA, IEEE Computer Society (2003) 204 5, 19, 26, 34

[27] Oppliger, R.: Security Technologies for the World Wide Web. Artech House, Inc., Norwood, MA, USA (2002) 5, 15, 18, 19, 21, 22, 23, 26, 30, 31, 32, 33, 34, 35, 40, 41, 107

[28] Park, J., Sandhu, R.: Secure cookies on the web. IEEE Internet Computing **4**(4) (2000) 36–44 5, 32, 34, 51, 107

[29] Mikko, H., Vuorimaa, P.: Secure Web Forms with Client-Side Signatures. In: ICWE. (2005) 340–351 5, 48, 49

[30] Deitel, H., Deitel, P., Nieto, T.: Internet and World Wide Web How to Program. Prentice Hall PTR, Upper Saddle River, NJ, USA (2004) 5, 14, 15, 16, 18, 19, 21, 33, 34

[31] Stein, L.: Web Security: A Step-by-Step Reference Guide. Addison-Wesley, Reading (1998) 5, 26, 34, 51

[32] Scott, D., Sharp, R.: Specifying and Enforcing Application-Level Web Security Policies. IEEE. Knowl. Data Eng **15**(4) (2003) 771–783 5, 34, 42, 50, 51, 60, 64, 67

[33] Kirkegaard, C., Moller, A.: Static analysis for Java Servlets and JSP. In: Proc. 13th International Static Analysis Symposium, SAS '06. Volume 4134 of LNCS., Springer-Verlag (2006) Full version available as BRICS RS-06-10. 6

[34] Acunetix: The importance of web application scanning (2005) http://www.sql-server-performance.com/wpaper_web_app_scanning.asp, Accessed Date: 20/4/2005. 7, 14, 29

[35] Tzay, J., Huang, J., Wang, F., Chu, W.: Constructing an Object-Oriented Architecture for Web Application Testing. IJ. Information Science and Eng. **18**(1) (2002) 59–84 7, 18, 34

[36] Boston Consulting Group: Report of the E-Business Opportunities Roundtable. Fast Forward: Accelerating Canada's Leadership in the Internet Economy (2000) http://www.e-com.ic.gc.ca/epic/site/ecic-ceac.nsf/en/h_gv00222e.html, Canada, Accessed Data: 5/12/2005. 7, 18, 32, 107

[37] Acunetix: Web applications: What are they? what of them?. (2007) http://www.acunetix.com/websitesecurity/web-applications.htm, Accessed Data: 15/2/2007. 7, 51

[38] Cardone, R., Soroker, D., Tiwari, A.: Using XForms to simplify web programming. In: WWW '05: Proceedings of the 14th international conference on World Wide Web, New York, NY, USA, ACM Press (2005) 215–224 7, 32, 107

[39] Erni, A., Norrie, M.: Approaches to accessing databases through web browsers (1998) A. Erni and M. C. Norrie. Approaches to Accessing Databases through Web Browsers. INFORMATIK, Journal of the Swiss Informaticians Society, October 1998. 7

[40] Lam, M.S., Martin, M., Livshits, B., Whaley, J.: Securing web applications with static and dynamic information flow tracking. In: PEPM '08: Proceedings of the 2008 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation, New York, NY, USA, ACM (2008) 3–12 7, 107

[41] Chong, J., Pal, P., Atigetchi, M., Rubel, P., Webber, F.: Survivability architecture of a mission critical system: The DPASA example. In: ACSAC

'05: Proceedings of the 21st Annual Computer Security Applications Conference, IEEE Computer Society, Washington, DC, USA, IEEE Computer Society (2005) 495–504 13, 39, 43

[42] Bishop, M.: What is computer security? IEEE Security and Privacy **1**(1) (2003) 67–69 IEEE Educational Activities Department, Piscataway, NJ, USA. 13, 39

[43] Berners-Lee, T.: WWW: past, present, and future. Computer **29**(10) (1996) 69–77 http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=539724. 14, 15

[44] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1 (1999) citeseer.ist.psu.edu/fielding97hypertext.html. 15, 19, 20, 21, 22, 23, 25, 45, 46, 50, 73, 92, 116

[45] Rosenfeld, L., Morville, P.: Information Architecture for the World Wide Web. O'Reilly & Associates, Inc., Sebastopol, CA, USA (2002) 15

[46] Wikipedia: Web content (2008) http://en.wikipedia.org/wiki/Web_content, Accessed Date: 5/1/2008. 15, 16

[47] Wikipedia: Dynamic web page (2008) http://en.wikipedia.org/wiki/Dynamic_web_page, Accessed Date: 5/1/2008. 16, 17

[48] Wong, B.: Sizing up Your Web Server. Sun World Online (1997) http://sunsite.uakom.sk/sunworldonline/swol-10-1997/swol-10-sizeserver.html. 16, 17, 172

[49] Kalbach, J.: Designing Web Navigation. O'Reilly Media, Inc, Sebastopol (2007) 16

[50] Brabrand, C., Moller, A., Schwartzbach, M.: The bigwig project. ACM Trans. Inter. Tech. **2**(2) (2002) 79–114 17, 49

[51] Steel, W.: Hints for web authors (2003) Mississippi Centre for Supercomputing Research (MCSR), University of Mississippi, USA, http://www.mcsr.olemiss.edu/ mudws/webhints.html. 18, 21, 22

[52] Brabrand, C., Moller, A., Ricky, M., Schwartzbach, M.I.: PowerForms: Declarative client-side form field validation. World Wide Web Journal **3**(4) (2000) 205–314 Kluwer. 18, 19, 48, 49, 50

[53] Thistlewaite, P., Ball, S.: Active FORMs. Computer Networks **28**(7-11) (1996) 1355–1364 18

[54] Liddle, S., Embley, D., Scott, D., Yau, S.: Extracting data behind web forms. In Proceedings of the Joint Workshop on Conceptual Modeling Approaches for E-business: A Web Service Perspective (eCOMO 2002), pages 38–49, Tampere, Finland, October 2002. (2002) 18, 34, 40

[55] Bosak, J.: XML, Java, and the future of the web. World Wide Web Journal **2**(4) (1997) 219–227 19

[56] Girgensohn, A., Lee, A.: Seamless integration of interactive forms into the Web. In: Selected papers from the sixth international conference on World Wide Web, Essex, UK, Elsevier Science Publishers Ltd. (1997) 1531–1542 19

[57] Dustin, E., Rashka, J., McDiarmid, D.: Quality web systems: performance, security, and usability. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002) 19

[58] Alameldeen, A., Martin, M., Mauer, C., Moore, K., Xu, M., Hill, M., Wood, D., Sorin, D.: Simulating a $2M commercial server on a $2K PC. Computer **36**(2) (2003) 50–57 20

[59] Honkala, M.: Web User Interaction a Declarative Approach Based on XForms. Technology, Department of Computer Science and Engineering - Helsinki University of Technology, Espoo, Finland (2007) ISBN 978-951-22-8566-2. 20, 49

[60] Borenstein, N., Freed, N.: Mime – Multipurpose Internet Mail Extensions (1993) RFC 1521 and RFC 1522. 21

[61] Moller, A., Schwartzbach, M.: Interactive Web services with Java (2002) BRICS, Department of Computer Science, University of Aarhus, Notes Series NS-02-1. Available from `http://www.brics.dk/~amoeller/WWW/`. 22, 23, 29, 32

[62] Prosise, J.: Microsoft.NET Web Forms (2002) http://www.informit.com/articles/article.asp?p=25939, Accessed Date: 12/12/2005. 23, 24

[63] DOD: Trusted Computer System Evaluation Criteria (TCSEC) (1985) Department of Defense Standard 5200.28-STD, (The Orange Book.). 26

[64] IBM: X-Force 2006 Trend Statistics (2007) http://www.iss.net/documents/whitepapers/X_Force_Exec_Brief.pdf, Accessed Date: 7/8/2007. 26

[65] CERT: CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests. http://www.cert.org/stats (2002) 26, 44

[66] Cooke, W.: Stupid JavaScript Security Tricks. In: Proc. 20th National Information Systems Security, Conference, sponsored by NIST and the National Computer Security Center, Baltimore, MD (1997) 116–127 http://csrc.nist.gov/nissc/1997/proceedings/116.pdf. 26, 28

[67] Sengupta, A., Mazumdar, C., Barik, M.S.: e-Commerce security - A life cycle approach. In: Commerce Security; Threats And Vulnerabilities; Security Engineering LifeCycle; Security Standards; IT Act. Volume vol.30., SADHANA (2005) p.119–140 27, 33, 44, 51

[68] Anderson, R., Lee, J.: Jikzi - a new framework for security policy, trusted publishing and electronic commerce. Computer Communications **23**(17) (2000) 1621–1626 27

[69] Anderson, R., Lee, J.: Jikzi: A new framework for secure publishing. In: Proceedings of the 7th International Workshop on Security Protocols, London, UK, Springer-Verlag (2000) 21–47 27

[70] Ye, E., Yuan, Y., Smith, S.: Web Spoofing Revisited: SSL and Beyond. Technical Report TR2002-417, Dartmouth College, Computer Science, Hanover, NH (2002) This TR supercedes TR2001-409. 27, 28

[71] Zhou, X., Huang, H.K., Lou, S.L.: Authenticity and integrity of digital mammography images. IEEE Trans. Med. Imaging **20**(8) (2001) 784–791 27

[72] McGraw, G., Morrisett, G.: Attacking malicious code: A report to the infosec research council. IEEE Software **17**(5) (2000) 33–41 29, 52

[73] Open Web Application Security Project: The Ten Most Critical Web Application Security Vulnerabilities. Version 1.1 (2003) 29, 30, 33, 51

[74] Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology, London, UK, Springer-Verlag (1996) 1–15 29

[75] Scott, D.: Abstracting Application-Level Security Policy for Ubiquitous Computing. PhD thesis, University of Cambridge (2004) http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-613.html. 29, 64, 65, 69, 118, 120

[76] Core: Netscape Communications Corporation: Core JavaScript guide (1998) Mountain View, CA., http://developer.netscape.com/docs/javascript.html. 29

[77] Spinellis, D.: Book review: Securing Java: Getting down to business with mobile code. ACM Computing Reviews **40**(8) (1999) 378–379 29

[78] Lewis, J.P., Neumann, U.: Performance of Java versus C++. Technical report, Computer Graphics and Immersive Technology Lab, University of Southern California (2003) Updated 2004, http://www.idiom.com/ zilla/Computer/javaCbenchmark.html. 29, 106

[79] The last Stage of Delirium Research Group-Poland: Java and Java Virtual machine security vulnerabilities and their exploitation techniques (2002) Poland, Black Hat Briefings, Singapore, http://www.blackhat.com/presentations/bh-asia-02/LSD/bh-asia-02-lsd.pdf. 30, 31

[80] Paul, N., Evans, D.: .NET security: Lessons learned and missed from Java. In: ACSAC '04: Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04), Washington, DC, USA, IEEE Computer Society (2004) 272–281 30, 33, 34, 40, 41, 107

[81] McGraw, G., Felten, E.: Securing Java: getting down to business with mobile code. John Wiley & Sons, Inc., New York, NY, USA (1999) 30

[82] Chen, E.: Poison Java. (1999) IEEE Spectrum. 31

[83] Netscape DevEdge: Client-Side JavaScript Guide: Electronic Book (1999) http://devedge-temp.mozilla.org/library/manuals/2000/javascript/1.3/guide/intro.html. 32

[84] Huang, Y., Huang, S., Lin, T., Tsai, C.: Web application security assessment by fault injection and behavior monitoring. In: WWW '03: Proceedings of the 12th international conference on World Wide Web, New York, NY, USA, ACM Press (2003) 148–159 33, 74

[85] Corsaire: A modular approach to data validation in web applications (2006) white paper. 33

[86] Yuen, P.: Practical Cryptology and Web Security. Pearson Education (2005) 35, 37, 45, 47, 60, 99

[87] Pavlou, K., Snodgrass, R.: Forensic analysis of database tampering. In: SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data, New York, NY, USA, ACM (2006) 109–120 35, 99

[88] National Institute of Standards: Secure hash standard. U.S. Department of Commerce FIPS PUB 180-2 (2002) 35, 65, 118, 120

[89] Bellovin, S.M., Rescorla, E.: Deploying a new hash algorithm. In: Proceedings of the Network and Distributed System Security Symposium, NDSS 2006, San Diego, California, USA, The Internet Society (2006) 35, 65, 118, 120

[90] Dittmann, J., Wohlmacher, P., Nahrstedt, K.: Using cryptographic and watermarking algorithms. IEEE MultiMedia **8**(4) (2001) 54–65 37

[91] Ricca, F.: Analysis, Testing and Re-Structuring of Web Applications. In: ICSM '04: Proceedings of the 20th IEEE International Conference on Software Maintenance, Washington, DC, USA, IEEE Computer Society (2004) 474–478 37, 42, 64

[92] Smith, S.: Outbound authentication for programmable secure coprocessors. In: ESORICS '02: Proceedings of the 7th European Symposium on Research in Computer Security, Springer-Verlag, London, UK, Springer-Verlag (2002) 72–89 41

[93] Park, J., Jayaprakash, G., Giordano, J.: Component integrity check and recovery against malicious codes. In: AINA '06: Proceedings of the 20th International Conference on Advanced Information Networking and Applications - Volume 2 (AINA'06), Washington, DC, USA, IEEE Computer Society (2006) 466–470 43, 69, 78, 80

[94] Knight, J., Sullivan, K.: Towards a definition of survivability. In: ISW 2000. Information Survivability Workshop. Third Information Survivability Workshop - ISW-2000. 'Research Directions and Research Collaborations to Protect the Global Information Society', Boston, MA (2000) 85–90 43

[95] Lipson, H., Fisher, D.: Survivabilitya new technical and business perspective on security. In: NSPW '99: Proceedings of the 1999 workshop on New security paradigms for ACM, New York, NY, USA, ACM (2000) 33–39 43

[96] Kent, S., Atkinson, R.: Security architecture for the internet protocol (1998) RFC Editor. 44

[97] McLeod, S., Cohen, M.: SSL vulnerabilities. (2002) In Australian Computer Emergency Response Team *AusCERT* conference. 44, 45

[98] Fu, K., Sit, E., Smith, K., Feamster, N.: Dos and don'ts of client authentication on the web. In: SSYM'01: Proceedings of the 10th conference on USENIX Security Symposium, Berkeley, CA, USA, USENIX Association (2001) 9–19 44, 45

[99] WANGHAN, M., LUNG, L., WESTPHALL, C., RAGA, J.: Integrating SSL to jacoweb security framework: Project and implementation. IEEE/IFIP International Symposium on Integrated Network Management (2001) 779792 44, 45

[100] Guest, S., Schnitzenbaumer, John, C.: Web Services Enhancements 1.0 and Java Interoperability. (2003) Microsoft working draft. http://msdn2.microsoft.com/en-us/library/ms996947.aspx#wsejavainterop2_topic2#wsejavainterop2_topic2. 45

[101] Shao, Z.: Security of the design of time-stamped signatures. J. Comput. Syst. Sci. **72**(4) (2006) 690–705 47, 48

[102] Rivest, R., Shamir, A.: Payword and micromint: Two simple micropayment schemes. In: Proceedings of the International Workshop on Security Protocols, London, UK, Springer-Verlag (1997) 69–87 48

[103] Wusteman, J.: Web forms: the next generation. Library Hi Tech **21**(3) (2003) 367 - 381 48, 49

[104] Ghosh, A., Swaminatha, T.: Software security and privacy risks in mobile e-commerce. Commun. ACM 44(2) (2001) 51–57 49

[105] Ricca, F., Tonella, P.: Analysis and testing of web applications. In: ICSE '01: Proceedings of the 23rd International Conference on Software Engineering, Washington, DC, USA, IEEE Computer Society (2001) 25–34 50, 72

[106] Halfond, G., Orso, A.: Preventing SQL injection attacks using AMNESIA. In: ICSE '06: Proceedings of the 28th international conference on Software engineering, ACM, New York, NY, USA, ACM (2006) 795–798 50, 74

[107] Wallach, D.: A New Approach to Mobile Code Security. PhD thesis, Princeton University, Department of Computer Science (1999) citeseer.ist.psu.edu/wallach99new.html. 53

[108] Glisson, W., McDonald, A., Welland, R.: Web engineering security: a practitioner's perspective. In: ICWE '06: Proceedings of the 6th international conference on Web engineering, New York, NY, USA, ACM Press (2006) 257–264 55

[109] SearchAppSecurity: Keep the bad guys out: Build security into the SDLC (2006) SearchAppSecurity.com. 55, 56

[110] Valdes, A., Almgren, M., Cheung, S., Deswarte, Y., Dutertre, B., Levy, J., Saïdi, H., Stavridou, V., Uribe, E.: An architecture for an adaptive intrusion-tolerant server. In Christianson, B., Malcolm, J.A., Roe, M., eds.: Security Protocols Workshop. Volume 2845 of LNCS. Springer Verlag (2002) 158–178 58, 59, 65, 66, 87, 99, 118, 120

[111] Porras, P., Neumann, P.: EMERALD: Event monitoring enabling responses to anomalous live disturbances. In: Proc. 20th NIST-NCSC National Information Systems Security Conference. (1997) 353–365 58

[112] Scott, D., Sharp, R.: Abstracting application-level web security. In: WWW '02: Proceedings of the 11th international conference on World Wide Web, New York, NY, USA, ACM Press (2002) 396–407 60

[113] Scott, D., Sharp, R.: Developing Secure Web Applications. IEEE Internet Computing **6**(6) (2002) 38–45 60

[114] Brogden, B.: Java Developer's Guide to Servlets and JSP. Sybex; Pap/Cdr edition (2000) 64, 65, 102

[115] Koh, H., Kung, S., Park, J.: The method to choose architectural approaches in the software architecture design phase. icita **01** (2005) 103–106 65

[116] X. Wang, Y.Y., Yu, H.: Finding collisions in the full SHA1. In: In CRYPTO'05. (2005) 65, 120

[117] Haleem, M.A., Mathur, C.N., Chandramouli, R., Subbalakshmi, K.: Opportunistic encryption: A trade-off between security and throughput in wireless networks. IEEE Transactions on Dependable and Secure Computing **4**(4) (2007) 313–324 66

[118] Aljawarneh, S., Laing, C., Vickers, P.: Security policy framework and algorithms for web server content protection. In: ACSF '07, Liverpool, UK, Liverpool John Moores University (2007) 67

[119] Aljawarneh, S., Laing, C., Vickers, P.: Verification of web content integrity: A new approach to protecting servers against tampering. In Merabti, M., ed.: PGNET 2007 The 8th Annual Postgraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting, Liverpool, UK, Liverpool John Moores University (2007) 75

[120] Merkle, R.: A certified digital signature. In: CRYPTO '89: Proceedings on Advances in cryptology, New York, NY, USA, Springer-Verlag New York, Inc. (1989) 218–238 77

[121] Snodgrass, R.T., Yao, S.S., Collberg, C.: Tamper detection in audit logs. In: VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases, VLDB Endowment (2004) 504–515 98

[122] Provos, N., McNamee, D., Mavrommatis, P., Wang, K., Modadugu, N.: The ghost in the browser analysis of web-based malware. In: HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets, Berkeley, CA, USA, USENIX Association (2007) 4–4 98, 155

[123] Wang, P.F., Li, J.P.: The one-way function based on computational uncertainty principle. The Computing Research Repository (CoRR) **abs/0711.3663** (2007) 99

[124] Mihçak, M., Venkatesan, R.: A perceptual audio hashing algorithm: A tool for robust audio identification and information hiding. In: IHW '01: Proceedings of the 4th International Workshop on Information Hiding, London, UK, Springer-Verlag (2001) 51–65 99

[125] Zoysa, K.D., Muftic, S.: Bi-directional web document protection system for serious e-commerce applications. IEEE/ICPPW **00** (2002) 11 103

[126] Amornsinlaphachai, P.: Updating Semi-Structured Data. PhD thesis, University of Northumbria, Newcastle, UK (2007) 106, 156

[127] Troesser, D.: Filters in the Servlet 2.3 API. Java News Brief **Year 2001 Issues** (2001) OCI Educational Services, Sun Authorized Java Center, http://www.ociweb.com/jnb/archive/jnbMay2001.html. 107, 108, 109

[128] Apache: The Apache Software Foundation, Apache Tomcat (1999-2007) http://tomcat.apache.org/. 113

[129] Mosberger, D., Jin, T.: httperfa tool for measuring web server performance. SIGMETRICS Perform. Eval. Rev. **26**(3) (1998) 31–37 130

[130] G, T., Abbott, Lai, K.J., Lieberman, M.R., Price, E.C.: Browser-based attacks on tor. In Borisov, N., Golle, P., eds.: Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007), Ottawa, Canada, Springer (2007) 155

[131] Razmov, V., Simon, D.: Practical automated filter generation to explicitly enforce implicit input assumptions. In: ACSAC '01: Proceedings of the 17th Annual Computer Security Applications Conference, IEEE Computer Society, Washington, DC, USA, IEEE Computer Society (2001) 347 160

[132] Brown, A.B., Seltzer, M.: Operating System Bench-marking in the Wake of Lmbench: A Case Study of the Performance of NetBSD on the Intel x86 Architecture. In: the 1997 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems. (1997) 214–224 http://www.eecs.harvard.edu/vino/perf/hbench/sigmetrics/hbench.pdf. 170

[133] Graham, J.: Web server sizing. Dell Power Solutions (2001) Issue 3, http://www.dell.com/content/topics/global.aspx/power/en/ps3q01_graham. 193